Projet M1 : Application P2P Hybride avec RMI

Applications Réparties 2008-2009
Université Paris VIII / Parcours SRM / M1

Sujet :

Le but de ce projet est d'implémenter une application de partage de fichiers basée sur l'architecture P2P hybride. Tout type de fichier doit pouvoir être partagé par un pair et un même fichier peut être disponible simultanément sur plusieurs pairs du réseau. Parmi les noeuds du réseau, on effectue alors la distinction entre :

- **L'annuaire P2P** : il est unique et son adresse doit être connue des pairs. Il répertorie l'ensemble des fichiers disponibles sur le réseau (mais ne les héberge pas) et conserve la liste des pairs en ligne à un instant donné.
- **Les pairs** : ils ont simultanément le rôle de client et de serveur de fichiers. Ils disposent d'une interface homme-machine (IHM) pour les utilisateurs leur permettant :
  - de configurer un dossier contenant les fichiers à partager,
  - de rechercher un fichier par son nom sur le réseau et de le télécharger,
  - de suivre le déroulement d'un téléchargement,
  - de configurer les différentes options et préférences du client.
  - de spécifier les informations de connexion au serveur (identification du client et adresse rmi du serveur)

Le téléchargement d'un fichier disponible sur plusieurs pairs et effectué par **segments de taille fixe**. On récupère les différents segments d'un même fichier sur plusieurs pairs et on reconstitue ensuite le fichier dans son intégralité en s'assurant de préserver son intégrité.

Détails d'implémentation :

Détails généraux :

L'implémentation doit être effectuée en Java grâce à l'API RMI/JRMP de Sun avec l'environnement de développement Eclipse v3.3 ou supérieure.

L'utilisation du chargement à distance de classes Java, tel que supporté par RMI/JRMP n'est pas obligatoire car il nécessite la mise en place d'un serveur web pour le téléchargement des classes via HTTP. Pour éviter cette problèmatique on s'assurera que les classes nécessaires au fonctionnement indépendant des pairs et de l'annuaires soient disponibles dans leurs classpaths respectifs. Dans un premier temps on peut s'en assurer en intégrant toutes les classes du projet dans le .jar d'un pair et dans celui de l'annuaire.

L'intégrité des fichiers partagés est assurée grâce à la notion de somme de contrôle ("checksum"). Lorsqu'un fichier est publié par un pair sur l'annuaire, il est associé à une somme de contrôle qui est calculée par le pair à partir des données du fichier, elle permet :

- De l'identifier de manière unique (indépendamment du nom du fichier).
- De s'assurer que tous les pairs qui offrent un même nom de fichier, offrent les mêmes données pour ce nom de fichier (mêmes données = mêmes sommes de contrôle).
- De s'assurer, une fois le fichier reconstitué après un téléchargement qu'il n'est pas corrompu (en comparant sa somme de contrôle à celle publiée dans l'annuaire).

Nous vous conseillons d'utiliser la classe java.util.zip.Adler32 car elle permet de calculer une somme de contrôle presque aussi fiable qu'un CRC32 mais de manière beaucoup plus rapide.

Le développement de l'annuaire et des pairs doit être effectué dans des espaces de nommages (packages) différent d'un même projet eclipse. Voici un exemple de hiérarchie qui vous est **fournie à titre indicatif** :

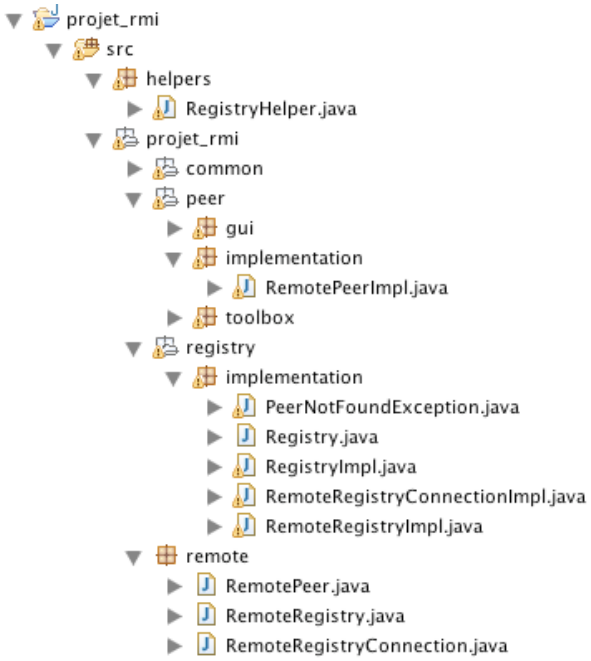

Figure 1 - Hiérarchie des packages et classes principales du projet

On peut voir dans la figure 1 que :

- L'ensemble des classes relatives à l'implémentation des pairs sont regroupées au sein d'un même package projet_rmi.peer
- L'ensemble des classes relatives à l'implémentation de l'annuaire P2P sont regroupées au sein d'un même package projet_rmi.registry
- Les interfaces "remote" des objets accessibles à distance par le protocole RMI/JRMP sont regroupées dans un même package externe à l'implémentation des pairs ou de l'annuaire. Il s'agit des classes RemotePeer, RemoteRegistry et RemoteRegistryConnection.

Détails pour l'annuaire :

Interface métier de l'annuaire : projet_rmi.registry.implementation.Registry
→ **implémentation** : projet_rmi.registry.implementation.RegistryImpl
Il s'agit de l'implémentation des fonctionnalités métier de l'annuaire (distinctes des préoccupations réseau) : enregistrement des clients, recherche de fichiers, ...

Interface Remote de l'annuaire : projet_rmi.remote.RemoteRegistry
→ **implémentation** : projet_rmi.registry.implementation.RegistryImpl
Il s'agit de la portion de l'annuaire accessible depuis le RMI Registry. L'interface n'offre qu'un seul point d'entrée aux clients : une méthode "logon" qui retourne à chaque client une connexion vers l'annuaire (instance de RemoteRegistryConnection).
Cette classe encapsule une instance RegistryImpl et possède une méthode main qui permet de lancer le serveur.

Interface Remote d'une connexion à l'annuaire : projet_rmi.remote.RemoteRegistryConnection
→ **implémentation** : projet_rmi.registry.implementation.RemoteRegistryConnectionImpl
Une connexion assure le dialogue réseau entre un pair donné et l'annuaire (publication des fichiers, recherche de fichiers, déconnexion,...). Il s'agit là aussi d'un objet RMI accessible à distance.

Attention de ne pas confondre l'annuaire de notre application P2P qui répertorie les pairs et les fichiers disponibles à tout instant sur le réseau (on parle aussi de "registry" en anglais) et le **RMI Registry** qui lui aura la charge de stocker une référence vers l'objet serveur de notre annuaire P2P de manière à ce qu'il soit visible aux pairs.

L'enregistrement de l'objet serveur d'annuaire P2P dans le RMI Registry permet d'obtenir une adresse RMI de la forme rmi:// adresse_machine/nom_de_l'annuaireP2P que les clients utiliseront pour accéder à l'annuaire P2P.

Afin de faciliter l'initialisation d'un RMI Registry directement depuis le code de votre annuaire P2P on vous fourni la classe RegistryHelper (cf. annexe 1).

Détails pour les pairs :

Interface Remote : projet_rmi.remote.RemotePeer
→ **implémentation** : projet_rmi.peer.implementation.RemotePeerImpl
Il s'agit d'un objet RMI accessible à distance qui contient l'ensemble des fonctionnalité métier et réseau d'un pair. Le code relatif à l'IHM doit se situer dans une classe séparée. Les pairs ne sont pas publiés dans le RMI Registry mais dans l'annuaire P2P (lui même accessible via le RMI Registry). Cette classe possède une méthode main qui permet de lancer le pair et son IHM.

On remarque donc que l'annuaire P2P n'est pas le seul composant de cette application à disposer d'une interface Remote. En effet, lorsqu'un pair veut télécharger un fichier il s'adresse directement aux autres pairs du réseau qui partagent ce fichier. D'où la nécessité pour chaque pair de disposer d'une interface sur laquelle il puisse être interrogé. Dans ce scénario, l'annuaire P2P sert à mettre les pairs en relation (il fournit la liste des pairs qui possèdent le fichier) **mais les données échangées entre les pairs ne transitent pas via l'annuaire** (architecture P2P hybride).

La mise au point d'une IHM est **obligatoire** pour les postes utilisateur (les pairs), facultative pour l'annuaire (on pourra se contenter d'une interface en ligne de commande). Les copies d'écran suivantes vous sont **fournies à titre indicatif** et vous êtes invités à développer une interface qui vous est propre, tout en conservant une certaines ergonomie :

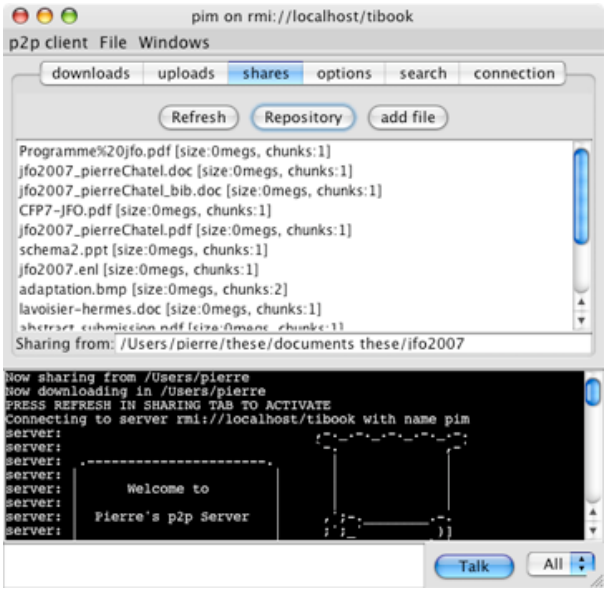

Figure 2 - Interface de partage de fichiers
(on peut aussi voir dans le bas en noir la "console" qui affiche les messages d'information émis par le serveur et une zone de "chat" entre les clients)

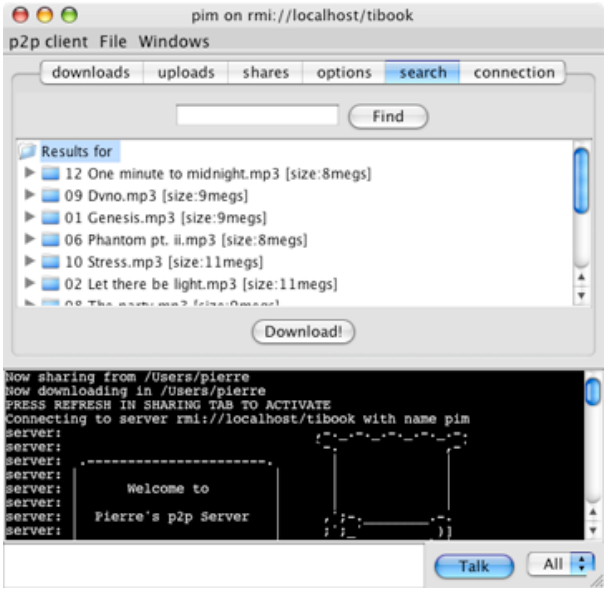

Figure 3 - Interface de recherche de fichier
(recherche de la chaîne vide affiche tous les fichiers disponibles)

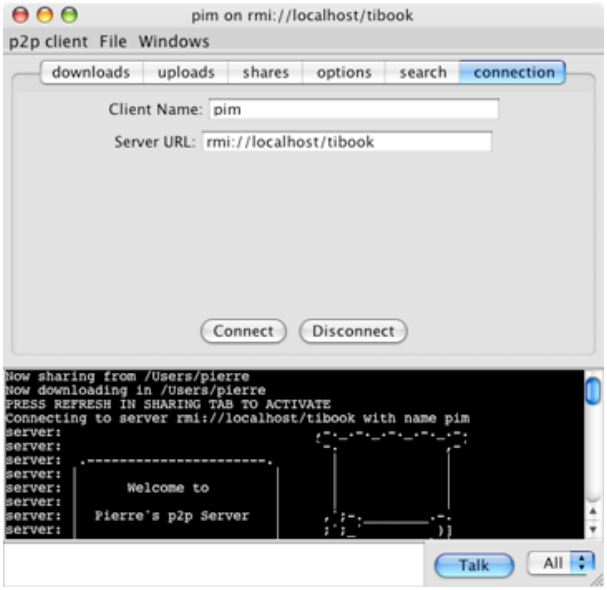

Figure 4 - Interface de spécification des informations de connexion au serveur

Architecture réseau de l'application :

La figure 5 présente simultanément l'architecture réseau des différents composants de cette application (pairs, RMI Registry, annuaire P2P) et certains des principaux échanges de messages entre ces composants. On peut voir la séquence d'actions nécessaires pour qu'un pair puisse télécharger un fichier sur d'autres pairs du réseau :

1. Le pair obtient une référence distante RMI sur l'annuaire P2P via le RMI Registry
2. Le pair se connecte sur l'annuaire et obtient une connexion.
3. Le pair utilise cette connexion pour obtenir la liste des pairs déjà enregistrés sur l'annuaire (étape 0) qui possèdent le fichier voulu.
4. Le pair d'adresse ensuite directement au(x) pair(s) qui possède(nt) ce fichier dans son intégralité (ou des morceaux recherchés du fichier).

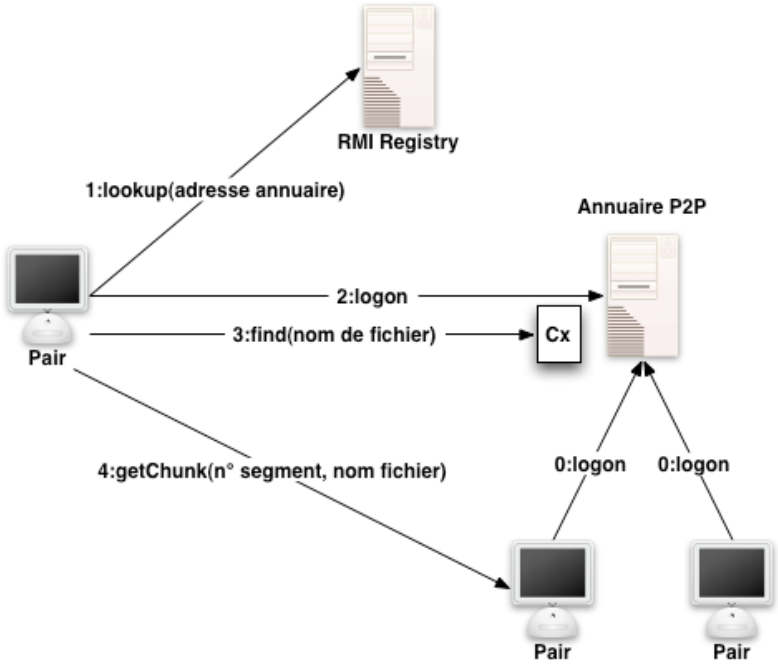

Figure 5 - Scénario de recherche de fichier (point de vue d'un pair)

Nota : il n'est pas nécessaire d'ouvrir une nouvelle connexion pour chaque recherche de fichier. Lorsqu'un client a obtenue une connexion il peut la conserver et l'utiliser pour tout autre action sur l'annuaire (publication de ses propres fichiers, nouvelle recherche, ...).

Rappel sur les architectures P2P hybrides:

Données distribuées mais index centralisé :

- On dispose d'un serveur central qui conserve des informations sur les pairs et peut répondre à des requêtes sur cette information (il joue souvent le rôle d'un annuaire de clients et de fi chiers).
- Les pairs sont responsables de l'hébergement des ressources partagées mais doivent indiquer leur disponibilité au serveur central.

Modalités :

Le projet doit être effectué par groupes de 3 étudiants (sans exceptions).

L'évaluation sera effectuée de deux façon :

- **A partir de code et de la documentation du projet** qui seront soumis par email au mois de Janvier 2009 (date précise à venir) à l'adresse suivante : chatelp@gmail.com. De plus, chaque soumission devra respecter la forme suivante :
  - Sujet : [Applications Reparties][M1][Groupe #x] Soumission
  - Corps du mail : noms des participants et indications éventuelles
  - Pièce jointe : une archive contenant le projet eclipse et toute la documentation (utilisateur et JavaDoc).
- **Sous forme d'une soutenance** en Janvier pendant laquelle chaque groupe de projet devra faire la démonstration du bon fonctionnement en réseau de son application. Il est fortement conseillé d'amener son propre matériel déjà configuré pour le fonctionnement en réseau et avec le projet déployé.

Principaux critères d'évaluation :

- **Respect des consignes et du sujet.**
- **Fonctionnement en réseau** : les clients et les serveurs doivent être situés sur des machines différentes montées en réseau local. Prévoir des machines pour la soutenance.
- **Simplicité de lancement de l'annuaire et des pairs** (double-click sur un .jar, script ant, uniquement depuis eclipse ... à vous de choisir).
- **Propreté du code** : utilisation de packages distincts pour les différents composants, documentation JavaDoc de toutes les classes (à ne pas confondre avec la documentation utilisateur, il s'agit ici d'une documentation pour le programmeur), formatage correct du code.
- **Documentation utilisateur du projet** : l'implémentation finale doit être fournie avec une documentation utilisateur qui présente les fonctionnalités et le mode d'utilisation de l'application.
- **Originalité de la réalisation** : essayez d'innover et d'ajouter quelques fonctionnalité originales à votre réalisation logicielle. Le succès d'une application se fait souvent sur les petits détails !

Annexe 1 : Classe RegistryHelper

```
/**
 * Cette classe facilite l'initialisation d'un registre RMI directement depuis
 * votre propre code. Il n'est donc pas necessaire de lancer le registre
 * separemment depuis la ligne de commande.
 *
 * Par contre, cela implique que le serveur RMI et son registre soient situees
 * sur la  machine.
 *
 * Ils partagent donc le meme classpath (car meme JVM) et toutes les classes
 * visibles par le serveur sont aussi visibles par l'annuaire. Il n'y a donc pas
 * de configuration fastidieuse du classpath de l'annuaire a effectuer.
 *
 * @author pierre
 *
 */
public class RegistryHelper {

	protected final static int DEFAULT_REGISTRY_PORT = 1099;

	/**
	 * settings of the registry
	 */
	protected int registryPortNumber = DEFAULT_REGISTRY_PORT;
	protected boolean shouldCreateRegistry = true;
	protected boolean registryChecked;

	public RegistryHelper() {
	}

	public int getRegistryPortNumber() {
		return registryPortNumber;
	}

	public void setRegistryPortNumber(int v) {
		registryPortNumber = v;
		registryChecked = false;
	}

	public boolean shouldCreateRegistry() {
		return shouldCreateRegistry;
```

```
}

public void setShouldCreateRegistry(boolean v) {
     shouldCreateRegistry = v;
}

/**
 * Methode principale qui permet d'initialiser le registre RMI pour
 * l'enregistrement d'objets serveurs.
 *
 * @throws java.rmi.RemoteException
 */
public synchronized void initializeRegistry()
          throws java.rmi.RemoteException {
     if (!shouldCreateRegistry)
          return; // don't bother
     if (registryChecked)
          return; // already done for this VM
     getOrCreateRegistry(registryPortNumber);
     registryChecked = true;
}

private static java.rmi.registry.Registry createRegistry(int port)
          throws java.rmi.RemoteException {
     return java.rmi.registry.LocateRegistry.createRegistry(port);
}

private static java.rmi.registry.Registry detectRegistry(int port) {
     java.rmi.registry.Registry registry = null;
     try {
          // whether an effective registry exists or not we should get a
          // reference
          registry = java.rmi.registry.LocateRegistry.getRegistry(port);
          if (registry == null)
               return null;
          // doing a lookup should produce ConnectException if registry
          // doesn't exist
          // and no exception or NotBoundException if the registry does exist.
          java.rmi.Remote r = registry.lookup("blah!");
          System.out.println("Detected an existing RMI Registry on port "
                    + port);
          return registry;
     } catch (java.rmi.NotBoundException e) {
          System.out.println("Detected an existing RMI Registry on port "
                    + port);
          return registry;
     } catch (java.rmi.RemoteException e) {
          return null;
     }
}

private static java.rmi.registry.Registry getOrCreateRegistry(int port)
          throws java.rmi.RemoteException {
     java.rmi.registry.Registry registry = detectRegistry(port);
     if (registry != null)
          return registry;
     // no registry created
     try {
          registry = createRegistry(port);
          System.out.println("Created a new registry on port " + port);
          return registry;
     } catch (java.rmi.RemoteException e) {
          // problem to bind the registry : may be somebody created one in the
          // meantime
          // try to find the rmi registry one more time
          registry = detectRegistry(port);
          if (registry != null)
               return registry;
          System.out
                    .println("Cannot detect an existing RMI Registry on port "
                              + port + " nor create one e=" + e);
          throw e;
     }
}
```

```
}
```