Projet M2 : Environnement collaboratif partagée

Applications Réparties 2008-2009
Université Paris VIII / Parcours SRM / M2

Sujet :

Le but de ce projet est d'implémenter un environnement collaboratif partagé, de manière à permette le travail à distance de plusieurs personne dans un mode de réunion de travail "dématérialisée".

Contrairement au projet p2p sur lequel vous avez travaillé en M1 l'année précédente, cette réalisation logicielle ne disposera pas d'un annuaire applicatif dédié autre que le registre RMI. Ainsi, **tous les "peer" sur le réseau sont équivalents** et disposent des fonctionnalités suivantes :

- Une zone de chat.
- Un tableau blanc virtuel.
- [BONUS] Un zone publique de partage de fichiers.

Ces fonctionnalités peuvent être alors utilisées dans le cadre d'une **session de travail**. Elle correspond à un regroupement de plusieurs peer au sein d'une réunion de travail :

- Une session est créée à l'initiative d'un peer A, pour participer à cette session un autre peer B du réseau doit connaître l'adresse RMI de ce peer A, ou de tout autre peer participant à la session (cf. Fig 1).
- En suivant la logique d'équivalence des peer évoquée ci-dessus, **tout peer doit pouvoir quitter une session (même le peer à l'initiative de la création de la session)** sans que cette dernière ne se termine pour autant.
- Un peer ne peut participer qu'à une seule session à la fois.
- La session cesse d'exister d'elle même lorsque plus aucun peer n'y participe.

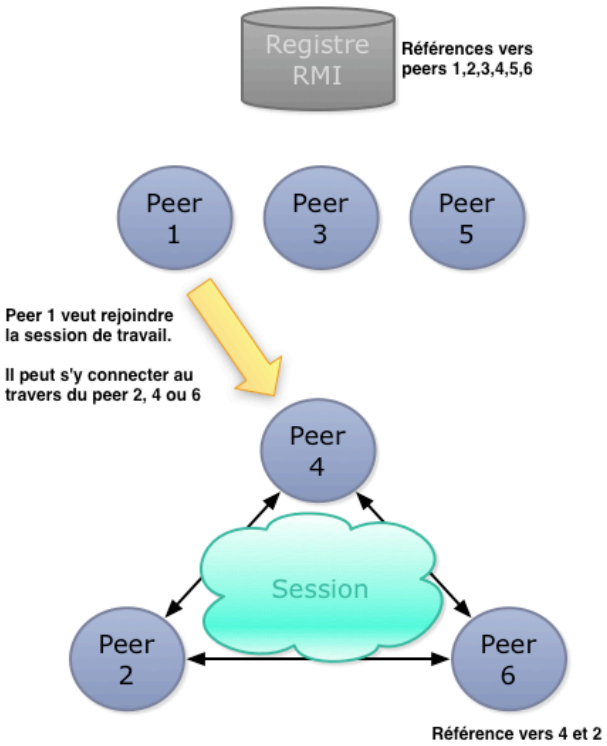

Fig 1 - Architecture globale et organisation d'une session

Outils et technologies à utiliser :

- Eclipse (>= v3.4.1).
- Java (>= v5).
- **RMI (cf. cours de rappel joint à ce projet).**
- Swing ou SWT pour l'interface.
- Projet indépendant de toute plate-forme : donc Linux, Windows ou MacOS X feront tous aussi bien l'affaire.

Retour sur les fonctionnalités de l'application :

1 - La zone de chat

Il s'agit d'une zone de chat partagée dans la mesure où tous les messages sont accessibles aux participants d'une session de travail. Pour ceux qui connaissent, pensez IRC plutôt que MSN.

Cf. Figure 2 pour un exemple possible d'interface de la zone de chat.

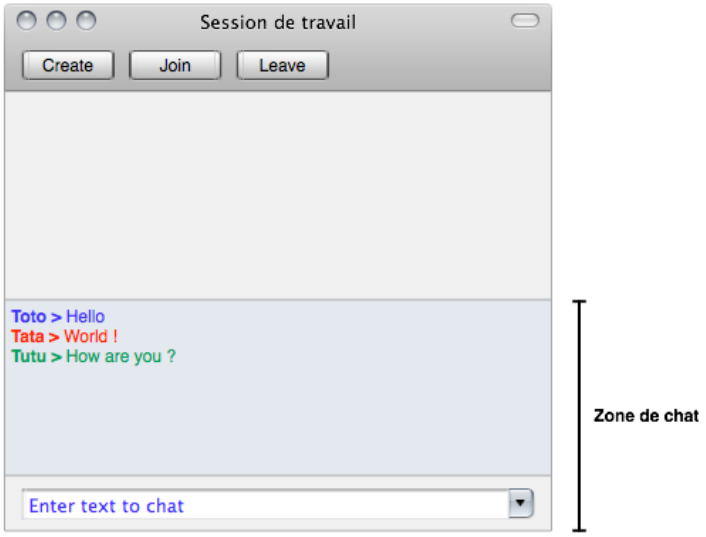

Fig 2 - Exemple d'interface pour le chat

2 - Le tableau blanc virtuel

Chaque peer dispose d'un affichage de ce tableau en local, mais le même contenu doit être affiché chez tous les participants.

- Ce tableau devra pouvoir contenir des tracés effectués à la souris [et éventuellement du texte].
- L'édition du tableau blanc virtuel est effectué **de manière concurrente** par tous les peers : cela signifie qu'ils peuvent tous y ajouter des informations simultanément.
- Afin de distinguer les informations de chaque participant à la session, on affectera **une couleur distincte à chaque peer** : par exemple, les tracés d'un peer A seront en bleu, alors que ceux d'un autre peer B en rouge.

Cf. Figure 3 pour un exemple possible d'interface du tableau blanc virtuel.

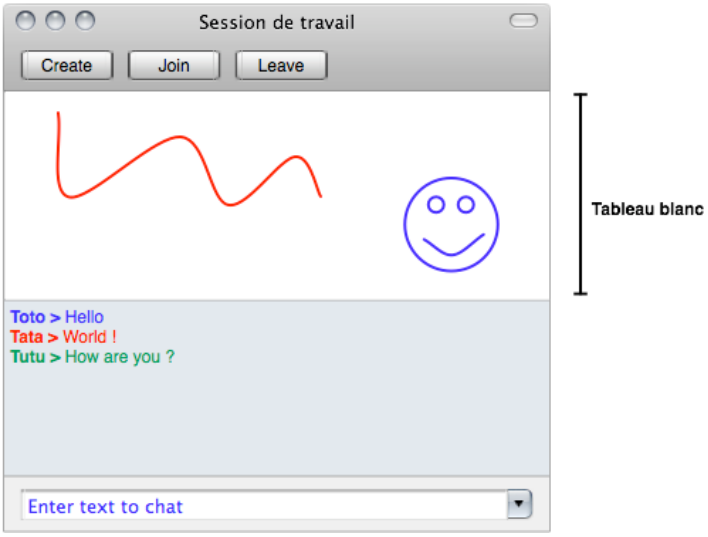

Fig 3 - Exemple d'interface pour le tableau blanc

3 - La zone publique de partage de fichiers

Chaque peer dispose d'une zone de partage de fichiers accessibles **aux autres peers de la session** à laquelle il participe, et uniquement à ces peers, pour des raisons évidentes de sécurité et de confidentialité.

Cf. Figure 4 pour un exemple possible d'interface pour le partage de fichiers.

Fig 4 - Exemple d'interface pour le partage de fichiers

Détails d'implémentation :

- Il n'existe qu'**un seul registre RMI pour toute l'application** dont la responsabilité est de garder une référence sur tous les peers du réseau, indépendamment des sessions de travail. Chaque peer devra donc s'y inscrire avec un nom qui lui est propre (méthode rebind du registry) de manière à être identifiable par les autres peers.
  - Il n'y a PAS besoin de développer un registre applicatif spécifique.
  - Afin de faciliter l'initialisation d'un RMI Registry directement depuis du code Java, on vous fourni la classe RegistryHelper (cf. annexe 1).
- Au sein d'une session de travail, tous les peers doivent partager la même information :
  - Quand un peer se connecte à une session, il récupère la liste de tous les participants, s'y ajoute, et communique aux autres participants cette modification.
  - Quand un peer envoi un message de chat, publie un nouveau fichier, ou modifie le tableau blanc, il envoie cette information à tous les participants (en parcourant sa liste locale) de manière à ce que tous les peers soient synchronisées (cf. figure 5).

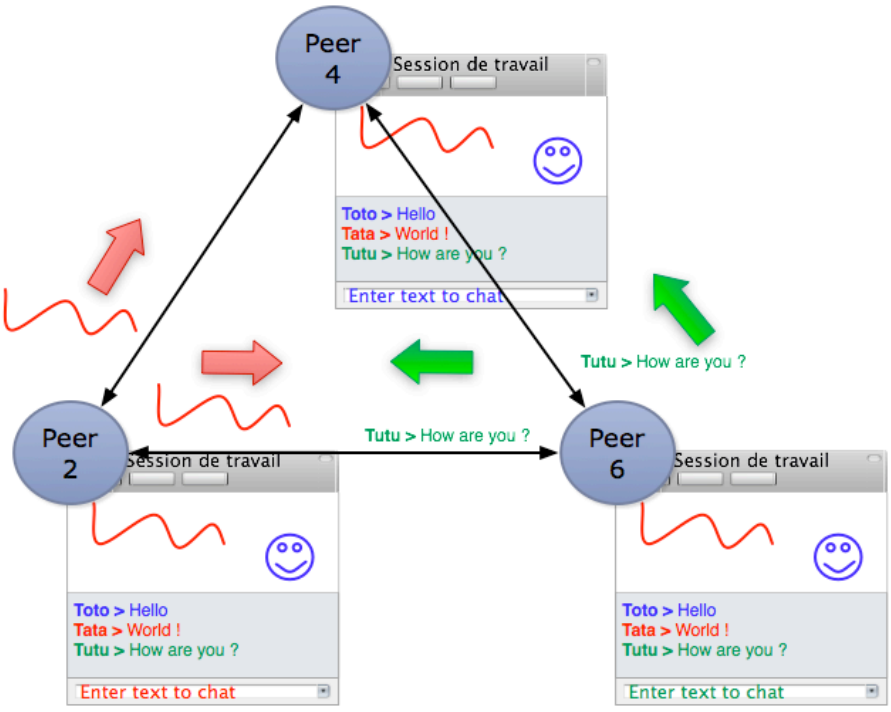

Fig 5 - Synchronisation des peers via RMI

En ce qui concerne l'implémentation de l'Interface Homme/Machine (IHM) de votre application, vous pourrez utiliser les bibliothèques SWING ou SWT "à la main" ou en utilisant avec un éditeur comme Jigloo (http://www.cloudgarden.com/jigloo/) permettant de construire votre IHM au sein même de Eclipse (cf. Figure 6).

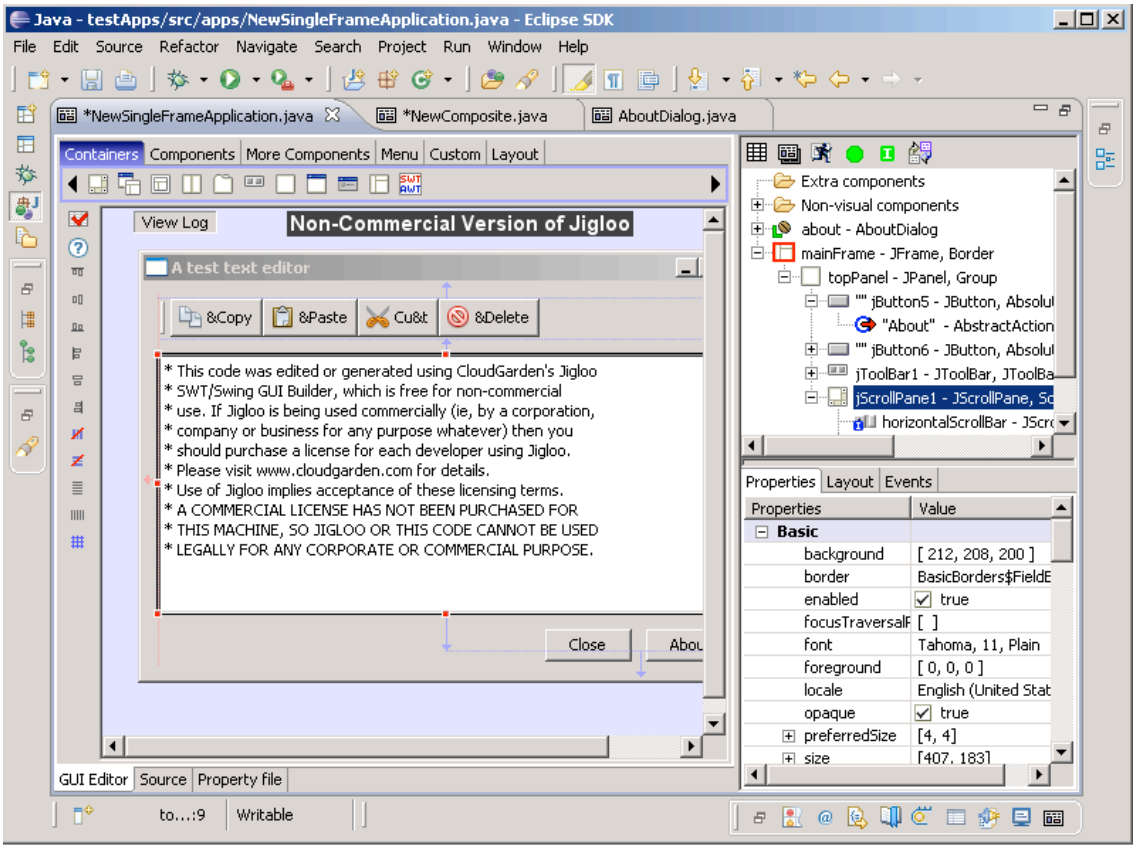

Fig 6 - Jigloo, l'éditeur d'interfaces graphiques SWING ou SWT pour Eclipse

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][M2][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 peers doivent être situés sur des machines différentes montées en réseau local. Prévoir des machines pour la soutenance.
- **Simplicité de lancement 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;
           }
     }
}
```