Université de Carthage

Institute National des Sciences Appliqués et de Technologie

Année Universitaire 2018-2019

COMPTE RENDU
Applications Réparties
TP2

Réalisée par :

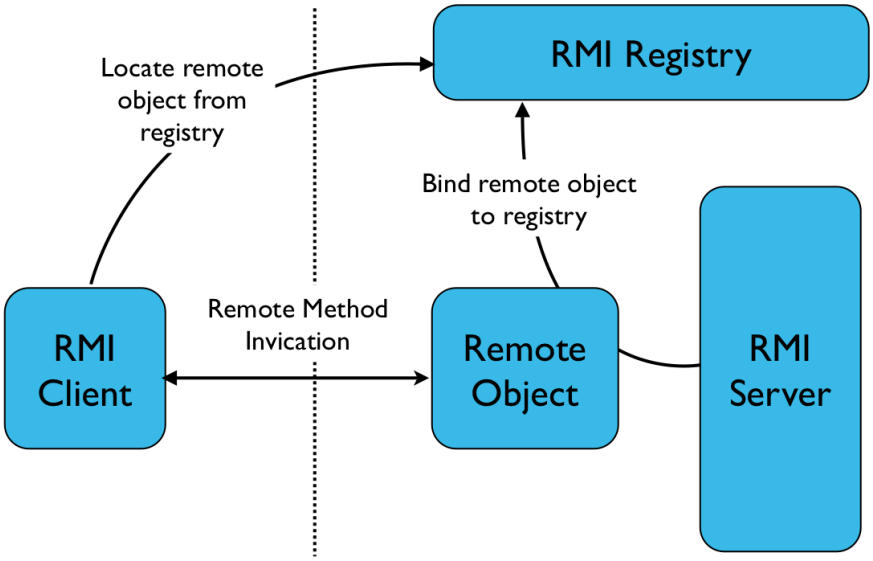
SNOUSSI Anis

BARGHOUDA Mohamed Lamine

L'ARCHITECRURE RMI

Introduction

Le but de RMI est de permettre l'appel, l'exécution et le renvoi du résultat d'une méthode exécutée dans une machine différente de celle de l'objet l'appelant. Cette machine peut être sur une machine différente pourvu qu'elle soit accessible par le réseau. La machine sur laquelle s'exécute la méthode distante est appelée serveur.



L'appel coté client d'une telle méthode est un peu plus compliqué que l'appel d'une méthode d'un objet local mais il reste simple. Il consiste à obtenir une référence sur l'objet distant puis à simplement appeler la méthode à partir de cette référence.

La technologie RMI se charge de rendre transparente la localisation de l'objet distant, son appel et le renvoi du résultat.

En fait, elle utilise deux classes particulières, le stub et le skeleton, qui doivent être générées avec l'outil **rmic** fourni avec le JDK.

Le stub est une classe qui se situe côté client et le skeleton est son homologue coté serveur. Ces deux classes se chargent d'assurer tous les mécanismes d'appel, de communication, d'exécution, de renvoi et de réception du résultat.

OBJECTIF DU TP

Architecture Cible du TP

Il s'agit de créer deux objets accessibles à distance :

- Un objet qui fait une addition
- Un objet qui calcule le nombre d'occurrence d'un caractère ou d'une chaine dans une autre chaine.

Les deux objets seront déployés sur une Machine M1 (la votre par exemple) et le client sera sur une machine M2 (celle de votre binôme par exemple) à condition de bien préciser son adresse IP dans le code du client. Bien entendu vous pouvez au départ faire le test sur une seule machine avec deux machines virtuelles distinctes comme expliqué dans la suite du TP.

Modélisation UML

Le développement se fera selon deux étapes :

1. Développement classique sans considérer les contraintes de l'accès distant
2. Développement prenant e compte les aspects liés à l'accès distant

Pour le développement classique on utilisera le diagramme UML ci-dessous :

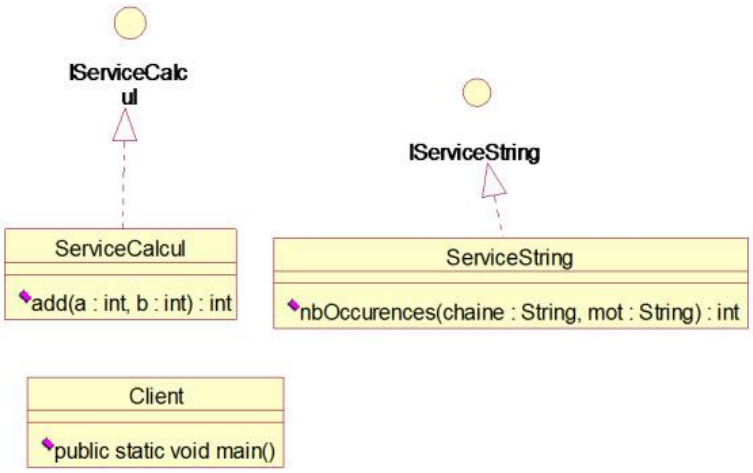


Figure 1 – Diagramme classes en mode local

Le développement avec accès distant se fera conformément au diagramme de classes ci- dessous.

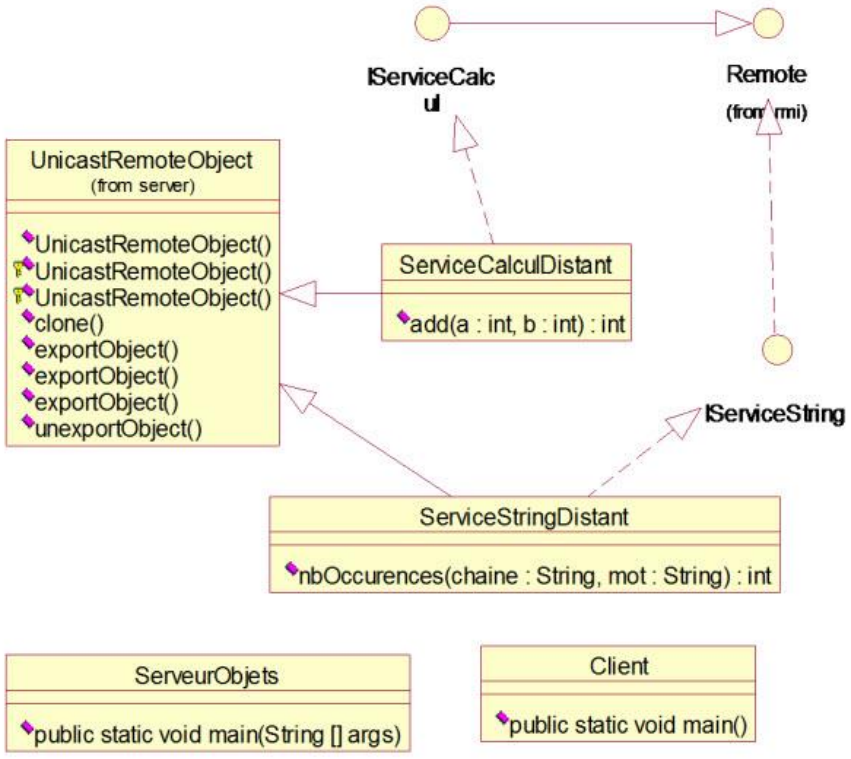


Figure 2 – Diagramme classes en mode Remote

Le client va utiliser des stub et skeleton pour envoyer des demandes à l'objet distant. Le stub et skeleton seront générés automatiquement.

Création de deux Objets « classiques » accessibles uniquement de manière Locale

Nous souhaitons disposer de deux services implantés sous forme de deux objets ServiceString et ServiceCalcul : Le premier réalise des opérations sur les chaînes de caractère, l'autre qui réalise du calcul numérique.

On va commencer par créer les trois classes conformément au diagramme de classe de la Figure 1

- L'interface **IServiceString** et la classe **ServiceString**
- L'interface **IServiceCalcul** et la classe **ServiceCalcul**
- Le client **ClientLocal**

```
public class ClientLocal{

public static void main(String [] args){

    ServiceString os = new ServiceString();
    ServiceCalcul oc = new ServiceCalcul();
    int i=os.NbOccurrences("h","hello world");
    int j =oc.add(2,5);
    System.out.println("hello world contient " + i + "fois \"h\"");
    System.out.println(somme de 2 et 5 = " + j);
   }
 }
```

ClientLocal.java

```
public interface IServiceString{
        public int NbOccurrences(String c, String mot);
}
```

IServiceString.java

```
public class ClientLocalclass ServiceString implements IServiceString {
 public int NbOccurrences(String c, String mot) {
  int longueur = mot.length();
  int Nb = 0;
  for (int i = 0; i < longueur; i++) {
   if ((mot.substring(i, i + 1)).equals(c))
    Nb++;
  }
  return Nb;
 }
}
```

ServiceString.java

```
public interface IServiceCalcul{
        public int add (int a, int b);
}
```

IServiceCalcul.java

```
public class ServiceCalcul{
        public int add (int a, int b) {
                return a+b;
        }
}
```

ServiceCalcul.java

Le résultat de l'exécution est :

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP/Manip 1 - Manière Local
$ javac *

Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP/Manip 1 - Manière Local
$ java ClientLocal
hello world contient 1 fois "h"
somme de 2 et 5 = 7
```

Modifications des deux Objets pour qu'ils fonctionnent à distance

On va maintenant se conformer au diagramme de classe de la Figure 2. Chaque méthode accessible à distance doit lever l'exception RemoteException.

1. Création de l'interface distante pour la classe ServiceString

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface IServiceString extends Remote {
 public int NbOccurrence (String c, String mot) throws RemoteException;
}
```

IServiceString.java

Remarque : Il est nécessaire que la méthode NbOccurrence doit lever l'exception RemoteException pour gérer les erreurs générées suite à un appel distant.

2. Modification de la classe ServiceString pour qu'elle fonctionne à distance

```
import   java.rmi.*;
import java.rmi.server.*;

 public class ServiceStringDistant extends UnicastRemoteObject implements
IServiceString {
  public ServiceStringDistant() throws RemoteException {
   super();
    }
    public int NbOccurrence (String c, String mot) throws RemoteException {
     int longueur=mot.length();
     int Nb=0;
     for (int i=0; i<longueur; i++) {
      if ((mot.substring(i, i+1)).equals(c)) Nb++;
     }
     return Nb;
      } }
```

3. Génération des amorces

La génération des amorces se fait grâce à la commande rmic (rmi compilateur).

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP/Manip 2 - Manière Distante
$ rmic ServiceStringDistant
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
```

4. Création de la classe Serveur

Pour que les objets soient accessibles à distance il faut une classe Serveur qui crée des instances et va les enregistrer auprès du serveur de nom (rmiregistry).

```
import java.rmi.*;
import java.rmi.server.*;

public class Serveur {
 public static void main(String[] args) {
  try {
   System.out.println("Serveur : Construction de l'implémentation");
   ServiceStringDistant ssd = new ServiceStringDistant();
   Naming.rebind("rmi://localhost:1099/Mot", ssd);
   System.out.println("ServiceString lié dans RMIregistry");
   System.out.println("Attente des invocations des clients ");
  } catch (Exception e) {
   System.out.println("Erreur de liaison de l'objet");
   System.out.println(e.toString());
  }
 }
}
```

Serveur.java

5. Démarrage de l'annuaire et du Serveur

Pour exécuter la classe client il faut d'abord lancer deux processus qui vont tourner constamment :

- Lancer une console *dos* (commande cmd). Démarrer le rmiregistry avec la commande : **rmiregistry**
- Lancer une console dos (commande cmd). Démarrer le Serveur avec la commande : **java Serveur**

6. Création et Exécution du client distant

```
import java.io.*;
import java.util.*;
import java.rmi.*;

public class ClientDistant {
        public static void main (String [] argv) throws IOException {
          int nb=0; String c, mot; c=argv[0]; mot=argv[1];
          try {IServiceString s= (IServiceString) Naming.lookup("rmi://localhost:1099/Mot");
               nb=s.NbOccurrence(c, mot);
               System.out.println(" Dans la phrase "+mot+", il y a "+nb+" occurrences de " + c);
                 }
                 catch (Exception e) {
                  System.out.println("Erreur d'acces a un objet distant");
                   System.out.println(e.toString());
                 }
        }
}
```

ClientDistant.java

On lance une troisième console dos et on exécute la classe ClientDistant :

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP/Manip 2 - Manière Distante
$ java ClientDistant a alpha
 Dans la phrase alpha, il y a 2 occurrences de a
```

1. RMI avec deux classes distantes

On modifie toutes les classes nécessaires pour qu'on puisse accéder à **ServiceCalcul** et **ServiceString** au même temps.

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface IServiceCalcul  extends Remote {
        public int add (int a, int b) throws RemoteException; ;
}
```

IServiceCalcul.java

```
import java.rmi.*;
import java.rmi.server.*;

public class ServiceCalculDistant extends UnicastRemoteObject implements
IServiceCalcul {

 public ServiceCalculDistant() throws RemoteException {
  super();
 }

 public int add(int a, int b) {
  return a + b;
 }
}
```

ServiceCalculDistant.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface IServiceString extends Remote {
 public int NbOccurrence (String c, String mot) throws RemoteException;
}
```

IServiceString.java

```
import java.rmi.*;
import java.rmi.server.*;

public class ServiceStringDistant extends UnicastRemoteObject implements
IServiceString {
 public ServiceStringDistant() throws RemoteException {
  super();
 }
 public int NbOccurrence(String c, String mot) throws RemoteException {
  int longueur = mot.length();
  int Nb = 0;
  for (int i = 0; i < longueur; i++) {
   if ((mot.substring(i, i + 1)).equals(c)) Nb++;
  }
  return Nb;
 }
}
```

ServiceStringDistant.java

```
import java.rmi.*;
import java.rmi.server.*;

public class Serveur {
 public static void main(String[] args) {
  try {
   System.out.println("Serveur : Construction de l'implémentation");
   ServiceStringDistant ssd1 = new ServiceStringDistant();
   Naming.rebind("rmi://localhost:1099/Mot", ssd1);
   ServiceCalculDistant ssd2 = new ServiceCalculDistant();
   Naming.rebind("rmi://localhost:1099/ADD", ssd2);
   System.out.println("ServiceString lié dans RMIregistry");
   System.out.println("Attente des invocations des clients ");
  } catch (Exception e) {
   System.out.println("Erreur de liaison de l'objet");
   System.out.println(e.toString());
  }
 }
}
```

Serveur.java

```
import java.io.*;
import java.util.*;
import java.rmi.*;

public class ClientDistant {
 public static void main(String[] argv) throws IOException {
  int occ = 0;
  String c = argv[0], mot = argv[1];
  int a, b, nb;
  a = Integer.parseInt(argv[2]);
  b = Integer.parseInt(argv[3]);
  try {
   IServiceString s = (IServiceString) Naming.lookup("rmi://localhost:1099/Mot");
   occ = s.NbOccurrence(c, mot);
   System.out.println(" Dans la phrase " + mot + ", il y a " + occ + " occurrences de " + c);
  } catch (Exception e) {
   System.out.println("Erreur d'acces a un objet distant");
   System.out.println(e.toString());
  }
  try {
   IServiceCalcul l = (IServiceCalcul) Naming.lookup("rmi://localhost:1099/ADD");
   nb = l.add(a, b);
   System.out.println(a + " + " + b + " = " + nb);
  } catch (Exception e) {
   System.out.println("Erreur d'acces a un objet distant");
   System.out.println(e.toString());
  }
 }
}
```

ClientDistant.java

On vérifie l'exécution :

```
Anis SNOUSSI@Anis-Desktop MINGW64 ~/Desktop/TP/Qs1
$ java ClientDistant a alpha 22 5
Dans la phrase alpha, il y a 2 occurrences de a
22 + 5 = 27
```

2. RMI en utilisant deux machines distantes

Pour deux machines virtuelles distantes et se trouvant sur deux ordinateurs connectés sur le même réseau, on doit modifier notre code source, et notamment le client. (Remplacer **localhost** par l'**IP** de la machine distante)

- Coté serveur :

```
Microsoft Windows [Version 10.0.17134.590]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Program Files\Java\jdk-10.0.2\bin>rmiregistry
```

On lance le **rmiregistry**

```
Microsoft Windows [Version 10.0.17134.590]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Program Files\Java\jdk-10.0.2\bin>java Serveur
Serveur : Construction de lâ??implÃ©mentation
ServiceString liÃ© dans RMIregistry
Attente des invocations des clients
```

Puis on lance le serveur

- Coté client :

```
Anis SNOUSSI@Anis-Desktop MINGW64 /c/Program Files/Java/jdk-10.0.2/bin
$ java ClientDistant a Java
Dans la phrase Java, il y a 2 occurrences de a
```

On fait appel à la méthode distante

3. Autres manières d'implémentation

1. Création et installation de Security Manager

On peut créer et installer un Security Manager dont l'objectif est de protéger l'accès aux ressources du système depuis un code téléchargé et exécuté dans un JVM non fiable.

```
if (System.getSecurityManager() == null) {
    System.setSecurityManager(new SecurityManager());
}
```

2. Enregistrement du Stub dans le Registry à partir du Main()

On peut créer le Stub de notre Objet et l'exporter vers l'RMI runtime dans la méthode Main() :

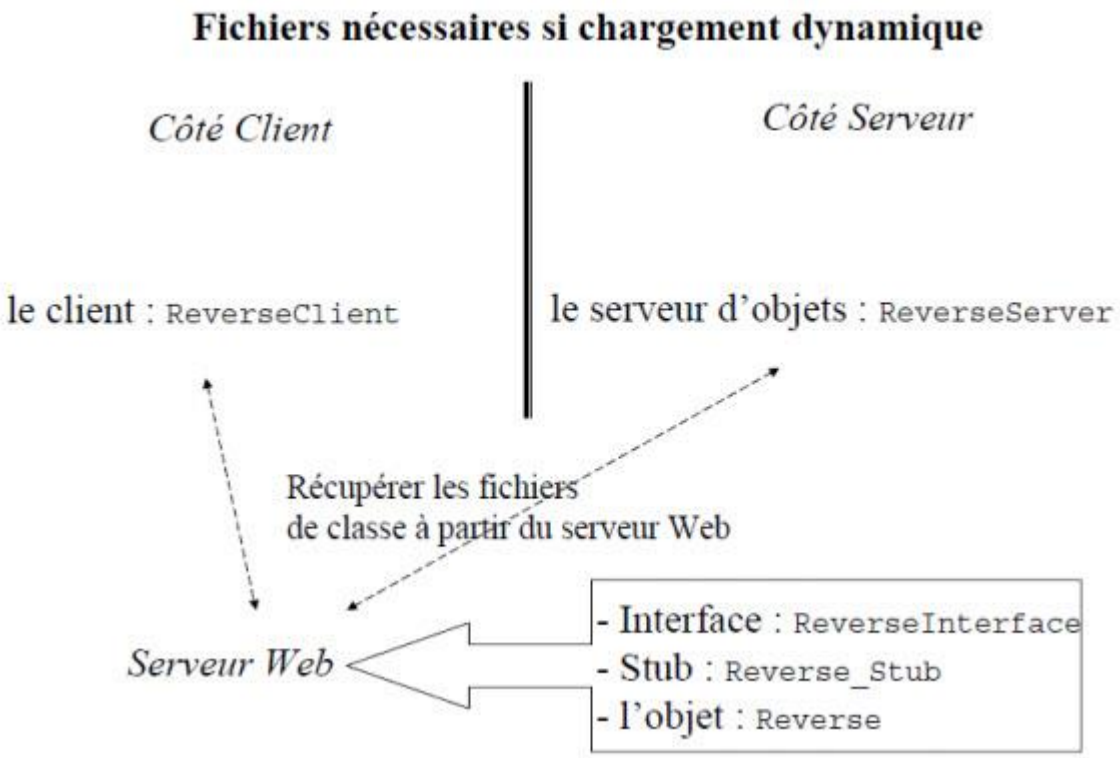
```
String name = "Mot";
Registry registry = LocateRegistry.getRegistry();
registry.rebind(name, stub);
```

Puis on ajoute le stub générer au Registry :

```
IServiceString engine = new ServiceString();
IServiceString stub = (IServiceString) UnicastRemoteObject.exportObject(ssd,0);
```

3. RMI Dynamique

Architecture du RMI Dynamique :



Etapes de réalisation :

- Reprendre les classes des deux composants répartis déjà réalisés dans la première partie RMI : ServiceString et ServiceCalcul.
- Modifier le Serveur pour qu'il devienne Dynamique : voir ce –dessous. Attention compléter ce code pour qu'il puisse aussi charger dynamiquement les classes de ServiceCalcul.
- Compiler toutes les classes et générer les amorces (rmic).
- Créer le fichier avec la police de sécurité "clien.policy" (voir ci-dessous). Son rôle est de permettre au classe Loader de RMI de se connecter à un serveur Distant.

- Lancer Netbeans. Sous l'onglet Serveur, cliquer sur serveur, ensuite click droit sur GlassFish serveur et choisir start. Le serveur http inclus dans GlassFish (Serveur http+ Serveur d'application –on le verra plus tard). Vérifier que le serveur a démarré en lançant un Navigateur avec l'url : **http://localhost:8080/.** La fenêtre de la Figure 1 apparaitre signifiant que le Serveur http est opérationnel.
- Toujours sous Netbeans, choisir l'onglet projets. Ensuite créer un nouveau projet exemple RMI du type Web Application et ce simplement pour bénéficier des possibilités du Serveur http. Choisir le Serveur et la version de java et cliquer sur Finish.
- Cliquer droit sur votre projet et choisir properties et noter l'emplacement de votre projet. Ensuite faites un expan (+) pour voir le contenu de votre projet vous remarquez le répertoire web pages ; C'est le répertoire ou doivent être sauvegardées les ressources publiques et disponible sur le web par le protocole http. Exemple avec l'URL **http://localhost:8080/RMI/index.html** saisie dans le navigateur c'est la page index.html du répertoire web pages qui sera lancée.
- Déplacer les Interfaces, Les deux composant ServiceString et ServiceCalcul ainsi que les deux amorces sous le répertoire Web Pages du projet RMI. Attention seul les .class sont nécessaires. Il faut aussi les supprimer du répertoire de travail courant pour être sûr qu'il va les chercher sous le serveur http.
- Lancer les commandes nécéssaire à savoir :
  - Lancer rmiregistry
  - Lancer le serveur : **c :>java -Djava.security.policy=client.policy -Djava.rmi.server.codebas=http://localhost :8080/RMI/ServeurDynamique**
  - Lancer enfin le client statique.

NB : Veuillez consulter le code source pour cette question