Professeur : Hamid Mcheick	Groupe : travail individuel
Trimestre : ÉTÉ2018	Date de distribution : 2 mai 2018
Pondération : 15 points	Date de remise : 16 mai 2018

Vous devez choisir une seule partie par les deux parties données ici-bas.

I – Première partie

Décrire le concept et l'architecture et donner un exemple (code) complet d'utilisation de l'un de deux protocoles de systèmes distribués suivants :
CoAP
ou
MQTT (ex : HiveMQ)

II - Deuxième partie

Le but du ce projet est de familiariser l'étudiant

1. au développement d'applications réparties en utilisant la communication par message (Socket), par procédure, par objet, etc.
2. aux concepts de passage de paramètres, de sérialisation, de réflexion, de nommage, de persistance, de collaboration dans les systèmes distribués, etc.

L'objective de ce travail pratique est l'implémentation d'une plateforme de calcule collaborative et distribuée. En effet, l'idée est de permettre à un client, avec ressources limitées (CPU, mémoire, etc.), l'exécution distribuée de certaines tâches quotidiennes. Pour ce faire, le client délègue l'accomplissement d'une ou plusieurs tâches à un serveur (ou machine) distant selon le diagramme ci-dessous :

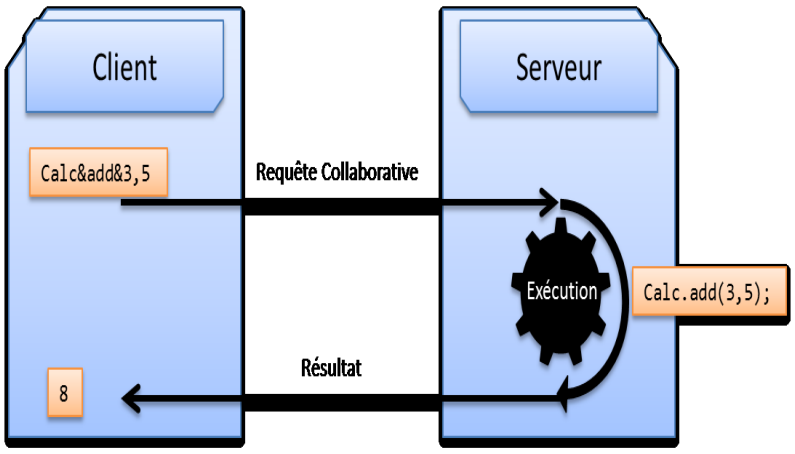

FIGURE 1 : PROCESSUS DU CALCULE DISTRIBUE

Le diagramme (Figure 1) illustre l'exécution d'une simple requête de calcule d'addition de deux nombres entiers. Ce processus évoque l'utilisation des Sockets ou autres techniques afin d'envoyer les requêtes (codes avec les données) à exécuter au serveur, ces requêtes comportent:

- Nom de la Classe et de la Méthode à invoquer, ainsi que les arguments de cette dernière.

- Fichier .java ou .class ou l'objet sérialisé à charger ou desérialiser lors de l'exécution.

Le client peut envoyer les requêtes au serveur en trois façons selon le protocole collaboratif suivant :

1) **SOURCEColl** : Code source de(s) la classe(s) demandé(es) ou
2) **BYTEColl** : Bytecode Code compilé de ces classes ou
3) **OBJECTColl** : Objets de ces classes

VOUS DEVEZ SEUELEMENT IMPLÉMENTER UN SEUL CHOIX PARMI CES TROIS FACONS.

Selon ton choix, ton serveur doit être capable de traiter la requête du client. Par exemple, si le client envoie des codes sources, alors le serveur doit être capable de

i) compiler ces codes sources
ii) charger (loader) ces codes
iii) Exécuter la (les) méthode(s) demandé(es)

Le serveur extrait les informations nécessaires (par exemple, classe en code octet-bytecode, nom de la classe, méthode, ses paramètres, ...), exécute le service demandé (méthode dans ce cas), et retourne le résultat au client, qui montre l'information dans son interface. Notons que vous avez besoin de charger la classe, exécuter la méthode (service) demandée dans cette classe, et retourner le résultat. J'ai ajouté deux classes (Calc.java et ByteStream.java) qui peuvent vous servir dans ce TP (voir pages suivantes).

Mots clés : Socket, Thread, InputStream, FileInputStream, Réflexion, OutputStream.
Livrables (en place): utilisez clé USB pour remettre ce TP1, qui sera corrigé en classe.

1. Code complet en Java avec commentaires 80%
2. Guide d'utilisation et trace d'exécution de l'application 20%

Remarque : Exécution et démonstration en classe par l'étudiant.

/** ---------- Calc.java ----------*/

```
package edu.uqac.gri.netreflect;

public class Calc {

	public int add(String a, String b){
		int x = Integer.parseInt(a);
		int y = Integer.parseInt(b);
		return x + y;
	}
}
```

/** ---------- ByteStream.java ----------*/

```
package edu.uqac.gri.netreflect;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

public class ByteStream {
      private static byte[] toByteArray(int in_int) {
              byte a[] = new byte[4];
              for (int i=0; i < 4; i++) {

                int  b_int = (in_int >> (i*8) ) & 255;
                byte b = (byte) ( b_int );

                a[i] = b;
               }
              return a;
            }

            private static int toInt(byte[] byte_array_4) {
              int ret = 0;
              for (int i=0; i<4; i++) {
                int b = (int) byte_array_4[i];
                if (i<3 && b<0) {
                  b=256+b;
                }
                ret += b << (i*8);
              }
              return ret;
            }

            public static int toInt(InputStream in) throws
java.io.IOException {
              byte[] byte_array_4 = new byte[4];

              byte_array_4[0] = (byte) in.read();
              byte_array_4[1] = (byte) in.read();
              byte_array_4[2] = (byte) in.read();
              byte_array_4[3] = (byte) in.read();

              return toInt(byte_array_4);
            }

            public static String toString(InputStream ins) throws
java.io.IOException {
              int len = toInt(ins);
              return toString(ins, len);
```

```
            }

            private static String toString(InputStream ins, int len)
throws java.io.IOException {
              String ret=new String();
              for (int i=0; i<len;i++) {
                ret+=(char) ins.read();
              }
              return ret;
            }

            public static void toStream(OutputStream os, int i) throws
java.io.IOException {
              byte [] byte_array_4 = toByteArray(i);
              os.write(byte_array_4);
            }

            public static void toStream(OutputStream os, String s) throws
java.io.IOException {
              int len_s = s.length();
              toStream(os, len_s);
              for (int i=0;i<len_s;i++) {
                os.write((byte) s.charAt(i));
              }
              os.flush();
            }

            private static void toFile(InputStream ins, FileOutputStream
fos, int len, int buf_size) throws
                  java.io.FileNotFoundException,
                  java.io.IOException {

              byte[] buffer = new byte[buf_size];

              int       len_read=0;
              int total_len_read=0;

              while ( total_len_read + buf_size <= len) {
                len_read = ins.read(buffer);
                total_len_read += len_read;
                fos.write(buffer, 0, len_read);
              }

              if (total_len_read < len) {
                toFile(ins, fos, len-total_len_read, buf_size/2);
              }
            }
```

```
    private static void toFile(InputStream ins, File file, int
len) throws
          java.io.FileNotFoundException,
          java.io.IOException {

      FileOutputStream fos=new FileOutputStream(file);

      toFile(ins, fos, len, 1024);
    }

    public static void toFile(InputStream ins, File file) throws
          java.io.FileNotFoundException,
          java.io.IOException {

      int len = toInt(ins);
      toFile(ins, file, len);
    }

    public static void toStream(OutputStream os, File file)
        throws java.io.FileNotFoundException,
               java.io.IOException{

      toStream(os, (int) file.length());

      byte b[]=new byte[1024];
      InputStream is = new FileInputStream(file);
      int numRead=0;

      while ( ( numRead=is.read(b)) > 0) {
        os.write(b, 0, numRead);
      }
      os.flush();
    }
  }
```