Université d'Evry Val d'Essonne, Master Informatique et Systèmes

Conception et Programmation d'Applications Réparties
M1/CPAR
TD 2 : Communication avec les Sockets

Tarek Melliti & Pascal Poizat

2011-2012

Exercice - 1 *Socket sychnrone*

Les sockets implantent un mode de communication asynchrone non-typé : l'envoie n'est pas bloquant (indépendamment du type de message envoyé) tandis que la réception est bloquante dans le cas où le tampon est vide mais ne l'est pas sur le type de message. Dans ce qui suit, on confondra le nom des messages et leurs type et comme en cours, l'action d'envoi d'un type de message a sera noté par $!a$ et la réception par $?a$.

Nous voulons mettre en place deux types de donnée qui remplacent les deux classes $ServerSocket$ et $Socket$ et qui assurent une communication synchrone typée :

- un processus qui veut émettre un message de type a (!a) sera bloqué tant qu'aucun processus n'est en état d'attente d'un message de même type ;
- Un processus en attente d'un message de type a ($?a$) est bloqué tant qu'aucun processus n'est en état de l'émettre.

Nous allons donc réaliser une solution en héritant des deux classes $ServerSocket$ et $Socket$ que nous allons nomer respectivement $SyncServerSocket$ et $SyncSocket$.

$SynchServerSocket$

Cette classe ne fait qu'accepter les demandes de connexion comme $ServerSocket$ et crée une instance de type $SynchSocket$.

$SynchSocket$

La classe $SynchSocket$ est une socket qui maintient une référence vers une instance de type $Socket$ et offre deux méthodes en plus :

- la méthode $send$ permet d'envoyer (selon la sémantique synchrone décrite plus haut) un type de message passé en paramètre ;
- la méthode $receive$ prend en paramètre une liste de type de message et implémente l'action de réception (selon la sémantique synchrone) de l'un des types de message qui se trouve dans liste en paramètre.

Voici le code de la classe $SynchServerSocket$ (complet) et $SynchSocket$ (incomplet).

```java
import java.net.*;
public class SyncServerSocket extends ServerSocket{

    public SyncServerSocket() throws IOException {
        // reprendre le même constructeur que ServerSocket
        super(); }

public SyncServerSocket(int a, int p) throws IOException {
    // reprendre le même constructeur que ServerSocket
    super(a,p); }
public SyncServerSocket(int a) throws IOException {
    // reprendre le même constructeur que ServerSocket
    super(a);}
public Socket accept() throws IOException{
    // créer une instance de SynchSocket en
    //lui passant la socket résultante d'une demande de connexion
    return new SynchSocket(super.accept());}
}
```

```java
import java.net.*;
import java.io.*;
import java.util.*;
public class SynchSocket extends Socket{
private Socket s;
BufferedReader in;
PrintWriter out;
public SynchSocket (Socket s) throws IOException
{this.s=s;
in=new BufferedReader(new InputStreamReader(s.getInputStream()));
out=new PrintWriter(s.getOutputStream());
}
public synchronized void  Send (String mess) throws IOException
{ /* A compléter */}

public synchronized String  receive (Vector<String> mess) throws IOException
{ /* A compléter */}

}
```

FIGURE 1 – Le code des deux classes $SynchServerSocket$ et $SynchSocket$

1- Complétez le code des deux méthodes $send$ et $receive$ de la classe $SynchSocket$ de telle façon que l'on aie une émulation de la communication synchrone.

Exercice - 2 *Système réparti communicant synchrone*

Soit un système réparti qui permet de réglementer l'accès aux soins aux employés d'une entreprise. Le système est composé de trois partenaires sur trois sites différents : l'Hôpital, le Travail et l'Employé. L'architecture du système est représentée par la figure 2 : Le système est composé de trois points d'accès (i) CH offerte par l'Hopital à l'Employé (ii) CA offerte par le Travail à l'Hôpital et (iii) CT offerte par le Travail à l'Employé.

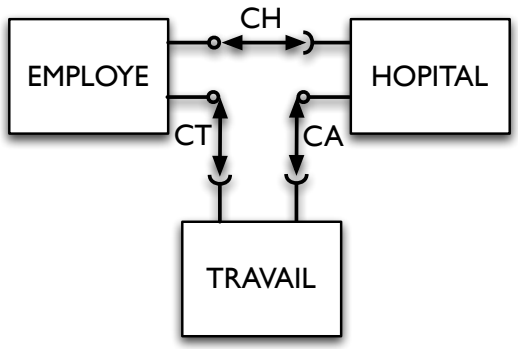

FIGURE 2 – L'architecture du système

Les trois partenaires interagissent en mode synchronne sur la base d'un protocole définie par trois LTSs communicants donnés dans la figure 3. Chaque type de message est précédé par le point d'accès sur lequel se passe la communication.

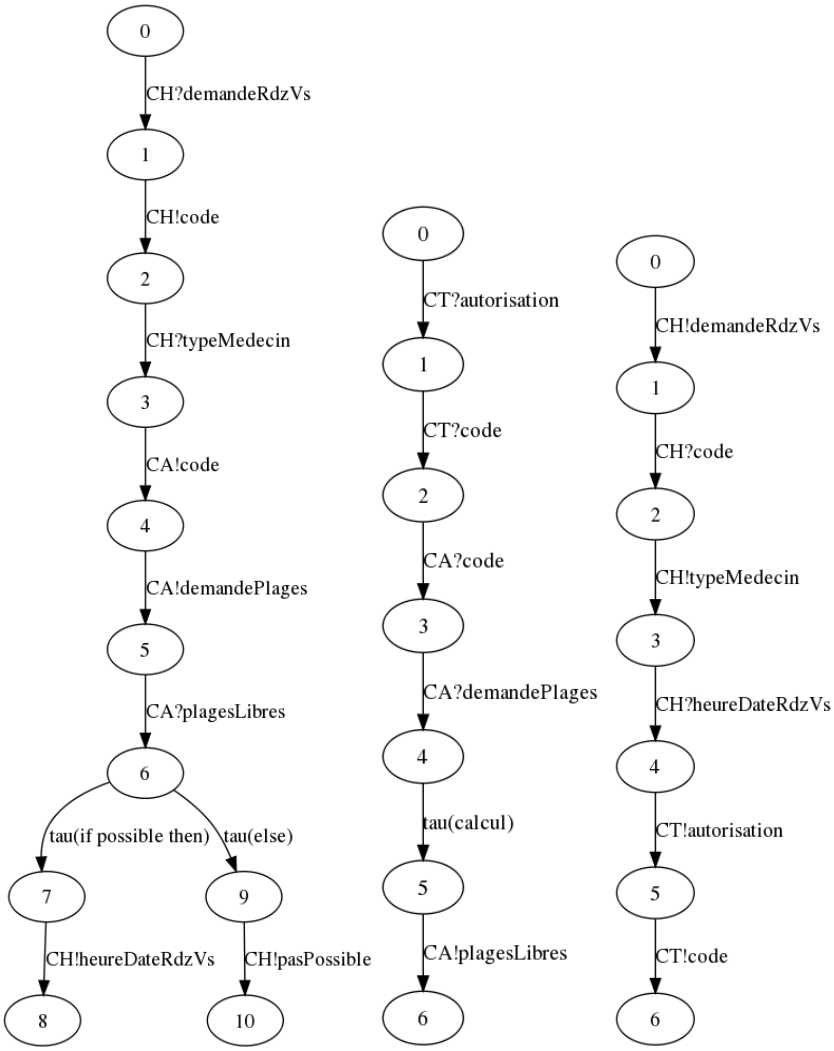

FIGURE 3 – Le LTS communicant de chaque partenaires : l'Hopital, Travail et Employé

1- Implantez le comportement de chaque partennaire en utilisant les $Threads$ ainsi que les deux classes $SynchServerSocket$ et $SynchSocket$.
2- Ecrivez trois $main$, une pour chaque partenaire, qui permettent de lancer une instance de chaque partenaire.
3- Est-ce le programme bloque ? Si oui, dites pouquoi et proposez une correction des protocoles.