Université d'Évry
Val d'Essonne

Master Informatique / MIAGE

Systèmes et Applications Répartis (SAR)

TD 2 : Communication synchrone avec les Sockets

Exercice 1 (Sockets synchrones)

Les sockets implantent un mode de communication asynchrone non-typé : l'envoi n'est pas bloquant (quelque soit le 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 et le type d'un message. L'envoi d'un type de message a sera noté $!a$ et la réception sera notée $?a$. Nous voulons mettre en place deux classes, remplaçant *ServerSocket* et *Socket*, et assurant 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 réaliser une solution en sous-classant *ServerSocket* et *Socket* par deux classes, respectivement *SyncServerSocket* et *SyncSocket*.

- *SyncServerSocket* : ne fait qu'accepter les demandes de connexion, comme *ServerSocket* et crée une instance de type *SyncSocket*.
- *SyncSocket* : maintient une référence vers une instance de type *Socket* et offre deux méthodes de 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 un type ou une collection de types de messages et implante l'action de réception (selon la sémantique synchrone) de l'un des types de message qui se trouve dans la collection en paramètre.

Voici le code incomplet des classes *SyncServerSocket* et *SyncSocket*.

Listing 1 – Classe SyncServerSocket à compléter.

```
1  public class SyncServerSocket extends ServerSocket{
2
3      public SyncServerSocket() throws IOException{
4          super();
5      }
6
7      public SyncServerSocket(int p, int b) throws IOException{
8          super(p, b);
9      }
10
```

```
11      public SyncServerSocket(int p) throws IOException{
12          super(p);
13      }
14
15      public SyncServerSocket(int port, int backlog, InetAddress bindAddr) throws
16          super(port, backlog, bindAddr);
17      }
18
19      public SyncSocket accept() throws IOException{
20          // à compléter...
21      }
22  }
```

Listing 2 – Classe SyncSocket à compléter.

```
1   public class SyncSocket extends Socket {
2
3       private Socket s;
4       private BufferedReader in;
5       private PrintWriter out;
6
7       public SyncSocket(Socket s) throws IOException{
8           this.s = s;
9           in = new BufferedReader(new InputStreamReader(s.getInputStream()));
10          out = new PrintWriter(s.getOutputStream(), true);
11      }
12
13      public SyncSocket(String host, int port) throws IOException{
14          this(new Socket(host, port));
15      }
16
17      public void send(String msg) throws IOException{
18          // à compléter
19      }
20
21      public void receive(String msg) throws IOException{
22          // à compléter
23      }
24
25      public String receive(Collection<String> msgs) throws IOException{
26          // à compléter
27      }
28  }
```

Question 1. Complétez le code des trois méthodes *accept*, *send* et *receive* des classes *SyncServerSocket* et *SyncSocket* afin d'avoir une émulation de la communication synchrone avec des sockets.

Exercice 2 (**Système distribué Communicant Synchrone**)

Soit le système distribué 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 distribué est représentée par la figure 1.

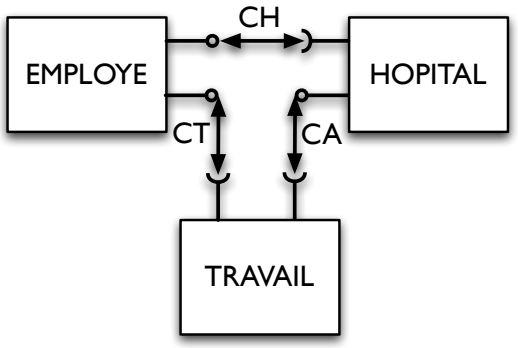

FIGURE 1 – L'architecture du système distribué.

Le système distribué est composé de trois partenaires et trois points d'accès (canaux/lignes de communication) :

— (i) CH offert par l'Employé à l'Hôpital ;
— (ii) CA offert par l'Hôpital au Travail ;
— (iii) CT offert par l'Employé au Travail.

Les trois partenaires interagissent en mode synchrone sur la base d'un protocole défini par trois LTSs (Système de Transition Labellisé) communicants donnés dans la figure 2.
Chaque type de message est précédé par le point d'accès sur lequel se passe la communication (canal synchrone).

Question 1. Est-ce que le système (programme) bloque ? Si oui, dites pourquoi et proposez une correction des protocoles.

Question 2. Implantez le comportement de chaque partenaire en utilisant les canaux de communication synchrones implémentés par la classe *SyncSocket*.

Question 3. Écrivez les trois partenaires : trois applications (3 *main*), une pour chaque partenaire, qui permettent de lancer une instance de chaque partenaire et de mettre en place l'architecture du système distribué souhaité.

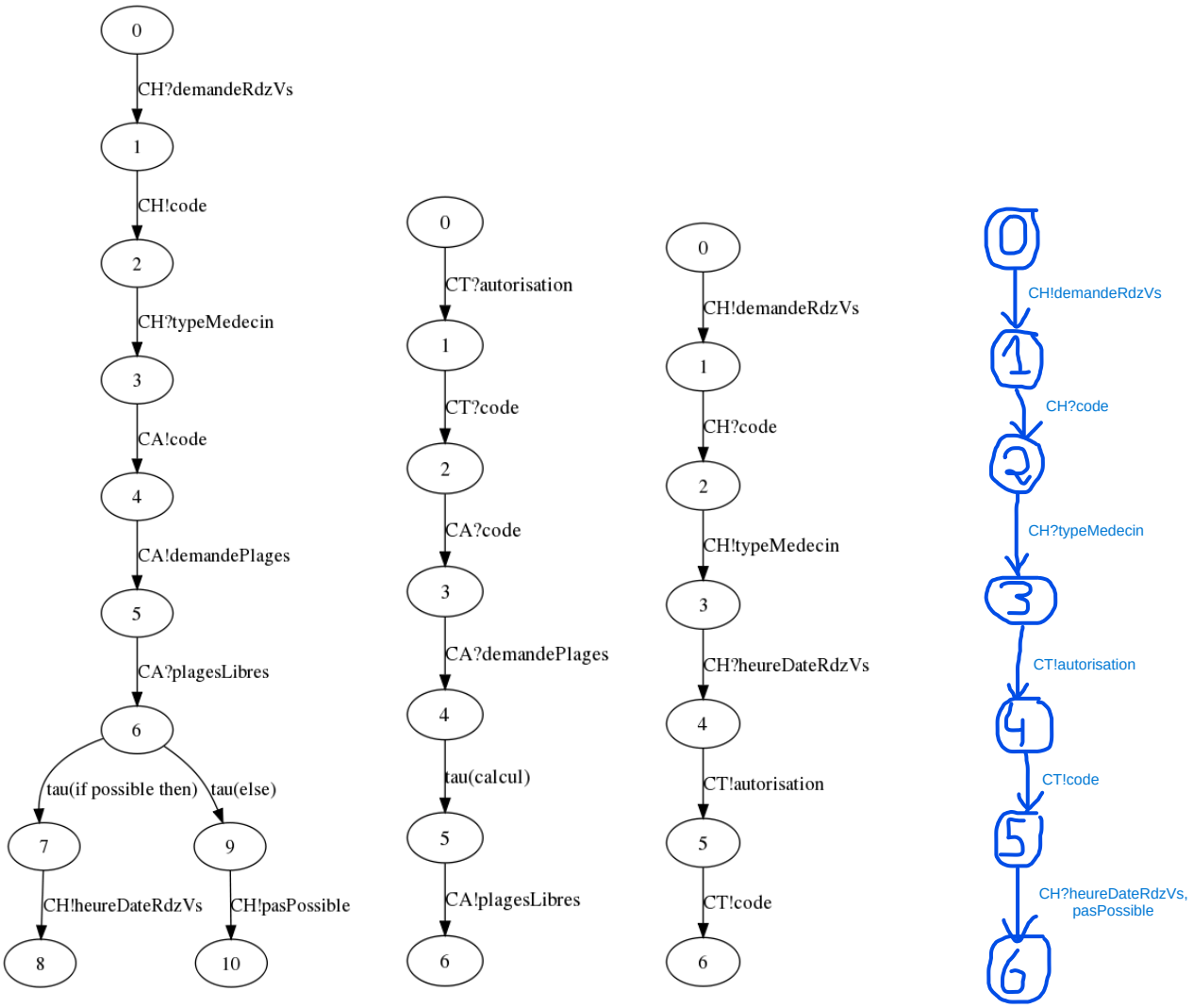

FIGURE 2 – LTS communicants des trois partenaires : Hôpital, Travail et Employé