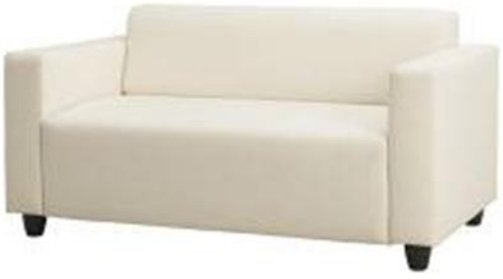
POLYTECH GRENOBLE

iSofa – Projet Innovant

Manuel Développeur



Flavien Garcia
Thibaud Michel

RICM4 2010-2011

Sensors

Pour interagir avec l'interface que nous avons développé, nous avons implémenté deux systèmes :

- Une interaction avec le clavier de l'ordinateur (utilisation d'un KeyListener)
- Une interaction avec la carte Arduino Mega (http://arduino.cc/en/Main/ArduinoBoardMega)

Une liaison par câble USB relie la carte au terminal. Les données sont envoyées sous forme de vecteur byte par la carte vers l'application grâce à la bibliothèque RXTX (http://rxtx.qbang.org) :

x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
PG	HG	BG	PD	HD	BD	Press AvG			Press ArG			Press AvD			Press ArD		

- PG = Bouton Poussoir Gauche (1 byte - boolean)
- HG = Bouton Haut Gauche (1 byte - boolean)
- BG = Bouton Bas Gauche (1 byte - boolean)
- PD = Bouton Poussoir Droit(1 byte - boolean)
- HD = Bouton Haut Droit (1 byte - boolean)
- BD = Bouton Bas Droit (1 byte - boolean)
- Press AvG = Capteur de pression Avant Gauche (3 bytes - int)
- Press ArG = Capteur de pression Arrière Gauche (3 bytes - int)
- Press AvD = Capteur de pression Avant Droit (3 bytes - int)
- Press ArD = Capteur de pression Arrière Droit (3 bytes - int)

La carte envoie les données en temps constant (environ 2 ms) pour faciliter les appuis simultanés. Le vecteur est parsé et distribué à des Sensors de type *com.res.Sensors.ButtonSensor* ou *com.res.Sensors.PressureSensor* qui sont stockés dans la *com.factories.SensorsFactory*.

Les capteurs sont distribués sous forme de pattern Observer/Observable. Une classe voulant être notifiée doit implémenter com.*res.Sensors.PressureSensorsListener* ou *com.res.Sensors. ButtonSensorsListener* et s'abonner soit aux boutons, soit aux capteurs de pression dans le *SensorsFactory*.

Notifications des boutons :

Les abonnés sont notifiés seulement quand le bouton change d'état sur le front montant.

Notifications des capteurs de pression :

Les données renvoyées par les capteurs sont traitées pour définir 4 états différents :

- Personne
- Une personne assise
- Une personne allongée
- Deux personnes

Les abonnées sont ensuite notifiées au changement d'état traité.

Notification par RMI :

De la même manière que les abonnements en interne, l'instance de "SensorsFactory" est partagée grâce à RMI. Toute classe peut alors s'abonner si elle implémente *ButtonSensorsListener* ou *PressureSensorsListener.* L'adresse de partage est `//localhost:4201/sf`

Voici un exemple d'utilisation :

```
1  package ext.rmi;
2
3  import java.rmi.RemoteException;
4  import java.rmi.server.UnicastRemoteObject;
5
6  import com.factories._SensorsFactory;
7  import com.res.sensors.ButtonSensorsListener;
8  import com.res.sensors.PressureSensorsListener;
9
10 public class TestRMI extends UnicastRemoteObject implements PressureSensorsListener, ButtonSensorsListener {
11     private static final long serialVersionUID = 3667359706807900229L;
12
13     protected TestRMI() throws RemoteException {
14         super();
15     }
16
17     public static void main (String[] args){
18
19         try {
20             TestRMI tr = new TestRMI();
21             _SensorsFactory sf = (_SensorsFactory) java.rmi.Naming.lookup("//localhost:4201/sf");
22             sf.subscribePermanentButtons(tr);
23             sf.subscribePermanentPressure(tr);
24         } catch (Exception e) {
25             e.printStackTrace();
26         }
27     }
28
29     @Override
30     public void event(int num, int key) {
31         String side, keyb;
32
33         if(num == ButtonSensorsListener.LEFT)
34             side = "Gauche";
35         else
36             side = "Droit";
37
38         if(key == ButtonSensorsListener.PUSH)
39             keyb = "Push";
40         else if(key == ButtonSensorsListener.UP)
41             keyb = "Haut";
42         else
43             keyb = "Bas";
44
45         System.out.println(side+" "+keyb);
46
47     }
48
49     @Override
50     public void event(int state) throws RemoteException {
51         switch(state) {
52             case PressureSensorsListener.BOTH :
53                 System.out.println("Deux");
54                 break;
55             case PressureSensorsListener.LAYED :
56                 System.out.println("Seul - Allongé");
57                 break;
58             case PressureSensorsListener.SIT_DOWN :
59                 System.out.println("Seul");
60                 break;
61             case PressureSensorsListener.NOBODY :
62                 System.out.println("Personne");
63                 break;
64         }
65     }
66 }
```

Interfaces (JPanel)

Toutes les interfaces sont stockées dans le package *com.Interface* et gérées dans *com.Interface.MainPanelInterface*. Pour rajouter une nouvelle interface il suffit de créer une nouvelle méthode dans *MainPanelInterface* et de l'ajouter au conteneur. Voici un exemple :

```
85      public void videos(){
86          videos = new VideosInterface(this, vf);
87          this.setPanel(videos);
88      }
```

Le MainPanelInterface est souvent passé en argument pour permettre de changer de menu en appelant la méthode du menu suivant.

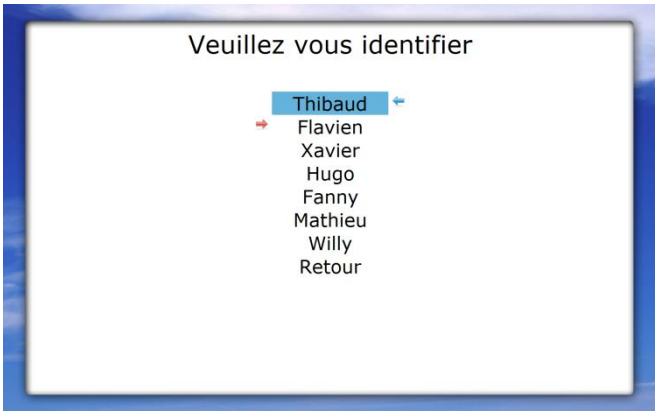
MyJList

Pour faciliter le développement de nouvelles interfaces et rester dans une charte graphique commune, vous pouvez utiliser ce JPanel qui est une JList avec curseurs. Voici le prototype :

```
public MyJList(String[] data, int nbPlayers, int size, int fontSize,
boolean ret, ArrayList<Icon> icons);
```

- data : correspond à la liste des données de JList
- nbPlayers : 1 ou 2 flèches
- size : largeur maximale des conteneurs (en pixels)
- fontSize : taille du font des conteneurs (en pixels)
- ret : si la liste doit comporter un retour
- icons : liste d'icones associées à la liste

Les 2 derniers paramètres sont optionnels.



Voici un exemple d'utilisation :

```
this.jl = new MyJList(data,1,350,48,false);
```

```
102     public void event(int num, int key) {
103         jl.event(num, key);
104         if(jl.isComplete()) {
105             SA.setNbPlayers(jl.getSelectedIndex1()+1);
106             SA.userList();
107         }
108     }
```

Jeux

Il existe à l'heure actuel 2 jeux, ils sont stockés dans le package *com.res.games* :

- PlayerReactor
- CrazyBuzz

Chaque jeu implémente une classe abstraite nommée *com.res. game*. Quand le jeu est lancé, il fait appel à la méthode start(). Un système de vote par accept (souvent utilisé avec le PushButton) peut être utilisé; il sert notamment à attendre la validation des 2 joueurs.

Chaque jeu comporte aussi une seule interface (JPanel) qui lui est propre, elle étend de GameInterface.