IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TD7/TP5 – Passage d'un modèle FSP au code Java

Modélisation et Programmation Concurrente – ELU 512

Objectifs

L'objectif de ces deux séances est de découvrir par la pratique comment produire un programme Java correspondant à un modèle FSP. En fin de séance, vous devriez être capable de :

- transformer un modèle simple en un programme Java,
- adapter cette transformation à vos besoins,
- savoir intégrer cette transformation dans votre processus de développement.

Il est important de noter que d'autres façons de construire le code Java sont possibles. Toutefois, plus le code Java est proche du modèle plus on peut espérer hériter une partie des bonnes propriétés du modèles.

Il est fortement conseillé de faire tous les exercices même s'ils n'ont pas pu être abordés en séance par manque de temps. En cas de problème, vous pouvez poser des questions sur le forum, contacter les enseignants, utiliser le monitorat informatique ou consulter la correction sur moodle.

Mise en pratique

Exercice 1

Soit le modèle FSP suivant :

```
CARPARK(N=4) = SPACES[N],
SPACES[i:0..N] = (
  when(i>0) arrive -> SPACES[i-1]
| when(i<N) depart -> SPACES[i+1]).
ARRIVALS   = (arrive -> ARRIVALS).
DEPARTURES = (depart -> DEPARTURES).
||CARPARK = (ARRIVALS || CARPARK(3) || DEPARTURES).
```

▷ **Proposer un code Java qui correspondrait à ce modèle.**

Exercice 2

Soit le modèle du télérupteur déjà vu :

```
BUTTON = (press -> BUTTON).
RELAY = (on -> off -> RELAY).
const N = 3
range Int = 1..N
||CONTACTOR = (b[i:Int]:BUTTON || RELAY)
    /{forall[i:Int] {b[i].press/{on,off}}}.
```

▷ **Proposer un code Java qui correspondrait à ce modèle. Pour rendre votre code plus visuel, vous pouvez utiliser la classe `Light` du paquet `display` qui fournit zone graphique (un `JPanel`) avec un dessin d'ampoule qui peut changer de couleur avec les méthodes `switchOn()` et `switchOff()`.**

Exercice 3

Dans le travail sur l'apprentissage de la modélisation en FSP, nous avons travaillé sur un sas. Nous allons reprendre cet exercice pour traduire le modèle obtenu en Java.

```
BUTTON = (press -> BUTTON).
DOOR = (open -> close -> is_closed -> DOOR).
CONTROLLER =
  (rcv_open -> ask_open -> conf_open -> open -> rcv_close -> close -> CONTROLLER).
||DOOR_SYS = (b[1..2]:BUTTON || CONTROLLER || DOOR)
/ {b[1..2].press / {rcv_open,rcv_close}}
@ {ask_open, conf_open, is_closed}.
SYNCHRO = CLOSED,
CLOSED = (
  ask_open[i:1..2] -> open[i] -> OPENED[i]
),
OPENED[i:1..2] = (
  ask_open[j:1..2] -> WAIT[i][j]
| is_closed[j:1..2] -> if (i==j) then CLOSED else OPENED[i]
),
WAIT[i:1..2][j:1..2] =(
  is_closed[k:1..2] -> if (i==k) then (open[j] -> OPENED[j]) else WAIT[i][j]
).
||SYS = (d[1..2]: DOOR_SYS || SYNCHRO)/{
       d[i:1..2].ask_open / ask_open[i],
       d[i:1..2].conf_open / open[i],
       d[i:1..2].is_closed / is_closed[i]
}.
```

▷ **Proposer un code Java qui correspondrait à ce modèle.**