Dr. Wolfgang Emmerich
Dept. of Computer Science
University College London
Pearson Building Room 402

8-Oct-99

Z01 Communications Software Engineering

Tutorial 2 - Work Sheet

Exercise 1: Modelling Sequential Process in FSP
A sensor measures the water level in a tank. The level (initially 5) is measured in units 0..9. The sensor outputs a low signal if the level is less than 2 and a high signal if the level is greater than 8, otherwise it outputs normal. Model the sensor as an FSP process. Exploit indexes and guarded actions for a concise process model.

Exercise 2: Translation between FSP and LTS
Transform the FSP description of the water level sensor that was developed in the above exercise into an equivalent Labelled Transition System.

Exercise 3: Modelling Concurrency in FSP
Extend the model of the client-server system described on Slides 3-12 and 3-13 such that there are three clients that use the server. Note that servers can only serve one client at a time. Validate your FSP model by drawing an equivalent LTS.

Exercise 4: Modelling Concurrency with Structure Diagrams and FSP
A museum allows visitors to enter through the east entrance and leave through its west exit. Arrivals and departures are signalled to the museum controller by the turnstiles at the entrance and exit. At opening time, the museum director signals that the museum is open and then both arrivals and departures are permitted. At closing time, the director signals that the museum is closed, at which point only departures are permitted. The museum exit closes when the last visitor has left the building. Given that there are four processes EAST, WEST, CONTROL and DIRECTOR, draw the Structure Diagram for the museum. Now provide an FSP description for each of the processes and the overall composition.

Dr. Wolfgang Emmerich
Dept. of Computer Science
University College London
Pearson Building Room 402

8-Oct-99

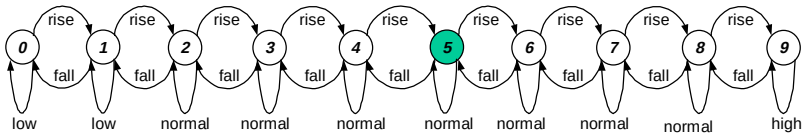
Z01 Communication Software Engineering

Answer Sheet

Exercise 1:

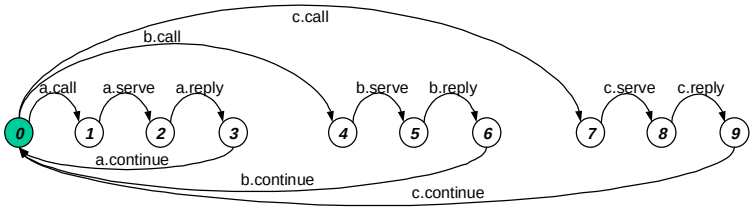
```
MEASURE(N=9) = MEASURE [5],
MEASURE[i:0..N] =
  ( when (i<2) low -> MEASURE[i]
  | when (i>8) high -> MEASURE[i]
  | when (i>1 && i<9) normal -> MEASURE[i]
  | when (i>0) fall -> MEASURE[i-1]
  | when (i<9) rise -> MEASURE[i+1]).
```

Exercise 2:

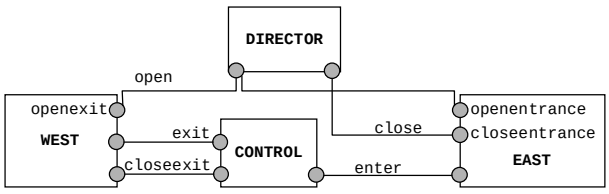


Exercise 3:

```
CLIENT = (call -> reply -> continue -> CLIENT)@{call,reply}.
SERVER = (call -> serve -> reply -> SERVER)@{call,reply}.
||CLIENTS_AND_SERVER =
  (a:CLIENT || b:CLIENT || c:CLIENT|| {a,b,c}::SERVER).
```



Exercise 4:




```
EAST = (openentrance-> EASTOPEN),
EASTOPEN = ( enter -> EASTOPEN | closeentrance -> EAST).
WEST = (openexit-> WESTOPEN),
WESTOPEN = ( exit -> WESTOPEN | closeexit -> WEST).
DIRECTOR = (open -> close->DIRECTOR).
CONTROL(N=100) = CONTROL[0],
CONTROL[i:0..N]= ( when (i==0) closeexit -> CONTROL[0]
                 | when (i>0) exit -> CONTROL[i-1]
                 | when (i<N) enter -> CONTROL[i+1]).
||MUSEUM = (EAST || WEST || DIRECTOR || CONTROL)
          /{open/openentrance, open/openexit, close/closeentrance}.
```