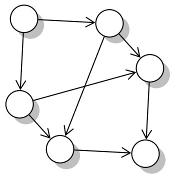

LINGI2143 — Concurrent Systems

Lab Session 2 – Modelling processes

S. Combéfis and C. Pecheur (September 29, 2011)

1 Composing processes

1. Start from the client-server example (File / Examples / chapter3 / client_server).

```
CLIENT = (call -> wait -> continue -> CLIENT).
SERVER = (request -> service -> reply -> SERVER).

||CLIENT_SERVER = (CLIENT || SERVER) / {call/request, reply/wait}.
```

   Modify it in order to model two new behaviours. The client has now a `timeout` that is started after a new `call` and if the timeout expires, the client is again ready to make a new `call`. The server can now have some `failure` that occurs whenever it receives a new `request`. If such a failure occurs, the `service` is not performed and the server is again ready to receive new `request`.

   (a) Build the parallel composition of the new `CLIENT` and `SERVER`.
   (b) The composition exhibits a deadlock. Can you identify in which situation (run) such a deadlock occurs ?
   (c) How can you modify the model(s) in order to avoid deadlocks ? Is your solution consistent with the real intuition of the CLIENT-SERVER model ? Explain.
   (d) Now, we would like to get an abstract version of the system, so, hide `timeout` and `failure`. What can you observe ?

2. A garbage dispatcher system is composed of several parts operating concurrently. There are two kind of garbage : `paper` and `other`. The dispatcher receives a garbage (`get.paper` or `get.other`) and can throw the garbage in the corresponding bin according to the type of garbage (`throw.paper` or `throw.other`). Before throwing a garbage in the corresponding bin, the dispatcher can empty the two bins (`empty`) simultaneously.

   The two bins have the same limited capacity (defined by a constant). A bin can accept a garbage (`throw.paper` or `throw.other` depending on the bin) whenever its maximal capacity is not reached. It can be emptied (`empty`) at any time.

   Model the garbage dispatcher system as an FSP process `SYSTEM` which is the composition of a `DISPATCHER` process, two processes for the bins (`PAPER_BIN` and `OTHER_BIN`) and a `USER` process which will feed the dispatcher with garbages. The alphabet should be:
   `{empty, {get, throw}.{other, paper}}`.

3. A drinks dispensing machine charges `15p` for a can of Sugarola. The machine accepts coins with denominations `5p`, `10p` and `20p` and gives change. Model the machine as an FSP process, `DRINKS`.

   [MK] exercise 2.5 (p.34)

   Consider the following variations. Can the machine distribute different drinks with different prices? Is there a limit on the number of coins the machine can keep before an order is made? Is it possible to model the amount of money (and the exact number of coins for each different value) that is inside the machine? Is it possible to model an algorithm so that the machine always gives back the minimal amount of the coins?

   Model a `USER` who wants a sugarola can and inserts two 20p coins. Examine the composed behaviour of the `DRINKS` machine with the `USER`. How can you model all the possible `USER`?

2 Bonus question

For each process you defined for the questions above, find the number of states and transitions of the corresponding LTS. If the process is parametrized, express the number of states and transitions in terms of the parameter.

[MK] refers to the reference book:

Concurrency: State Models & Java Programs by Jeff Magee & Jeff Kramer.