Shared Resource Code Generation

1 JCSP Code Templates

1.1 OnDemand

```
1  public class SRNameCSP implements CSProcess {
2
3    private final List<Any2OneChannel> splittedChannels;
4
5    private Any2OneChannel getChannel(String method, T1 a1,..,Tk ak){...}
6
7    /* WRAPPER IMPLEMENTATION */
8    public Tj methodj(arg0,..,argn) {
9      //@ assume PREj(arg0,..,argn);
10     One2OneChannel innerChannel = Channel.one2one();
11     getChannel(methodj,argl,..,argr).out().write(
12                          new Request(innerChannel, argl,..,argr));
13     //@ assume CPREj(arg0,..,argn);
14     // data to be returned
15     return ((Tj) innerChannel.in().read());
16   }
17
18   /* SERVER IMPLEMENTATION */
19   /** Constants representing API method's */
20   ...
21   private static final int METHODiXl = 0;
22   ...
23
24   public void run() {
25     /**
26       * One entry for each associated predicated.
27       * Union of all channel lists.
28       */
29     Guard[] inputs={methodiXlChannel.in(),...};
30
31     /**
32      *  Conditional reception for fairSelect().
33      *  Should be refreshed every iteration.
34      */
35     //@ assert inputs.length == K+splittedChannels.size();
36     boolean syncCond[]= new boolean[K+splittedChannels.size()];
37     < initialized syncCond >
38
39     final Alternative services = new Alternative(inputs);
40     int chosenService;
41
42     /** Server loop */
43     while (true) {
44       // refreshing synchronization conditions
45       < updating syncCond >
46       /*@ assume (\forall int i; i> = 0 i < syncCond.length;
47         @                 syncCond[i] ==> channelAssocCpre(i))
48         @*/
49
50       chosenService = services.fairSelect(syncCond);
51       /*@ assume chosenService < guards.length &&
52         @       chosenService >= 0 && syncCond[chosenService] &&
53         @       guards[chosenService].pending() > 0;
54         @*/
55
56       switch(choice){
```

```
        ...
        // method's request processing
        case METHOD_i X_l :
          //@ assert P_i && C_i(X_l);
          // if it is needed to pass spare information
          // this channel must be used for that
          Request request = ((Request)
                          getChannel(method_j,x_l).in().read());
          eid = innerMethod_i();
          request.getChannel().out().write(eid);
          break;
      }
    } // end while
  }// end run
}
```

1.2 Deferred Request

```
public class SRNameCSP implements CSProcess {

  /* WRAPPER IMPLEMENTATION */

  private final Any2OneChannel method_0Channel;
  ...
  private final Any2OneChannel method_NChannel;
  // variable declaration for inner state of the resource
  ...
  // method's wrapper schema
  public T_i method_i(T_1 arg_0,..,T_m arg_m) {
    //@ assume P && I
    One2OneChannel innerChannel = Channel.one2one();
    method_iChannel.out().write(
                    new Request(innerChannel,< footprint >));
    //if double send
    //@ assert P && I && C;
    innerChannel.out.write(...);
    T_1 value = (T_1) innerChannel.in().read();
    //@ assert Q && I;
    return value
  }

  // method accessing/modifying shared resource's inner state
  protected T_i innermethod_i(T_1 arg_1,..,T_m arg_m) {
    //@ assume P && C && I;
    S;
    //@ assert Q && I;
  }

  /* SERVER IMPLEMENTATION */
  ...
  private static final int METHOD_1 = 0;
  ...
  private static final int METHOD_N = N;
  ...
  private final Queue<Request> method_iRequests;
  ...
  public void run() {
    Guard[] inputs={method_iChannel.in(),...,method_NChannel.in()};
    Alternative services = new Alternative(inputs);
    int choice = 0;
    while (true) {
      choice = services.fairSelect();
      /*@ assume chosenService < guards.length &&
        @        chosenService >= 0 &&
        @        guards[chosenService].pending() > 0;
        @*/
      switch(choice){
        ...
        case METHOD_i :
          //@ assume P
          method_iRequests.add((Request) method_iChannel.in());
```

```
54            break;
55          ...
56        }
57        boolean requestProcessed = true;
58        while (requestProcessed) {
59          requestProcessed = false;
60          for all requests list do {
61            int queueSize = method_kRequests.size();
62            for (int i = 0; i < queueSize; i++) {
63              Request request = method_kRequests.poll(QUEUE_HEAD);
64
65              if (condition_k (request.getFootprint()) ) {
66                //@ assume I && condition_k ==> C ;
67                ChannelInput chIn = request.getChannel().in();
68                T values = (T)chIn.read();
69                results= this.innerMethod_k(values);
70                //@ assume I && Q ;
71                request.getChannel().out.write(results);
72                requestProcessed = true;
73              } else {
74                method_kRequests.offer(request);
75              }
76            }
77          }
78          //@ ensures there is no stored thread in any request list which its synchronization ↩
                condition holds
79        }
80      } // end while
81    }// end run
82  }
```