Shared Resource Code Generation

1 JCSP Code Templates

1.1 OnDemand

```
public class SRNameCSP implements CSProcess {

  private final List<Any2OneChannel> splittedChannels;

  private Any2OneChannel getChannel(String method, T1 a1,..,Tk ak){...}

  /* WRAPPER IMPLEMENTATION */
  public Tj methodj(arg0,..,argn) {
    //@ assume PREj(arg0,..,argn);
    One2OneChannel innerChannel = Channel.one2one();
    getChannel(methodj,argl,..,argr).out().write(
                          new Request(innerChannel, argl,..,argr));
    //@ assume CPREj(arg0,..,argn);
    // data to be returned
    return ((Tj) innerChannel.in().read());
  }

  /* SERVER IMPLEMENTATION */
  /** Constants representing API method's */
  ...
  private static final int METHODiXl = 0;
  ...

  public void run() {
    /**
      * One entry for each associated predicated.
      * Union of all channel lists.
      */
    Guard[] inputs={methodiXlChannel.in(),...};

    /**
     *  Conditional reception for fairSelect().
     *  Should be refreshed every iteration.
     */
    //@ assert inputs.length == K+splittedChannels.size();
    boolean syncCond[]= new boolean[K+splittedChannels.size()];
    < initialized syncCond >

    final Alternative services = new Alternative(inputs);
    int chosenService;

    /** Server loop */
    while (true) {
      // refreshing synchronization conditions
      < updating syncCond >
      /*@ assume (\forall int i; i> = 0 i < syncCond.length;
        @                 syncCond[i] ==> channelAssocCpre(i))
        @*/

      chosenService = services.fairSelect(syncCond);
      /*@ assume chosenService < guards.length &&
        @       chosenService >= 0 && syncCond[chosenService] &&
        @       guards[chosenService].pending() > 0;
        @*/

      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());
```

```
          break;
        ...
      }
      boolean requestProcessed = true;
      while (requestProcessed) {
        requestProcessed = false;
        for all requests list do {
          int queueSize = method_k Requests.size();
          for (int i = 0; i < queueSize; i++) {
            Request request = method_k Requests.poll(QUEUE_HEAD);

            if (condition_k (request.getFootprint()) ) {
              //@ assume I && condition_k ==> C ;
              ChannelInput chIn = request.getChannel().in();
              T values = (T)chIn.read();
              results= this.innerMethod_k (values);
              //@ assume I && Q ;
              request.getChannel().out.write(results);
              requestProcessed = true;
            } else {
              method_k Requests.offer(request);
            }
          }
        }
        //@ ensures there is no stored thread in any request list which its synchronization ↩
            condition holds
      }
    } // end while
  }// end run
}
```