Alpen-Adria Universitaet Klagenfurt
Mobile Systems Group
Univ.-Ass. DI Udo Schilcher

PR Mobile and Wireless Systems
Problems part 2

In the following Problems you will learn how to use the process and node model editors to create own models and use them in a simulation.

1. **Answer the following questions:**

   (a) For what purpose can a process model be used?

   (b) Which elements are part of a process model and what is their purpose?

   (c) What is the difference between a forced and an unforced state?

   (d) What is a logical Rx/Tx association in a node model good for?

2. **Create the models for a simple hub.**

   (a) Create a packet format.
   To create a new (simple) packet format, select File/New... from the menu and choose packet format. With the packet format editor create a packet containing the following fields:

Field	Type
dest_address	integer (32 bit)
id	integer (32 bit)

   (b) Create a process model for a hub.
   A process model consists of a FSM (finite state machine) and some code extensions written in C. For the simple hub functionality it is necessary to create a forced init state and an idle state. The init state is used to initialize all state variables needed and the idle state contains several transitions to itself to process all events. If you think that more states are needed, feel free to add more of them.

   Transitions can be unconditioned and conditioned. If you want to create a conditioned transition (e.g. which is only used when a packet arrives), you can enter it in the condition attribute of the transition. In general conditions are defined as macros inside the 'header block'. If you want to check, for example, if a packet has been received, you can use the following macro:

```
#define PK_ARRVL ( op_intrpt_type() == OPC_INTRPT_STRM )
```

In the executive attribute you can enter a C function call. The function has to be defined in the 'function block'. The first function you will have to write should forward all receceived packets to the appropriate output stream. Here we use a very simple address concept where the address in the packet directly maps to the number of the output stream. A fragment of the function is given, you only have to insert the two main function calls (a list of useful functions can be found at the end of this document):

```
static void route_pk()
{
  int dest_address;
  Packet *pkptr;
  FIN( route_pk() );
  pkptr = op_pk_get( op_intrpt_strm() );

  // INSERT: get the destination address
  // INSERT: send the packet
  FOUT;
}
```

The FIN and FOUT directives are very important to avoid that two tasks execute the code between these two statements at the same time (→ semaphore).

If a conditioned transition is used it is often important to add another transition with the condition 'default' to catch all other events not handled by one of the conditioned transitions.

(c) Create a node model for a hub.
Now you have to create a node model for the hub. A hub normally consists of one processor plus several point-to-point receiver/transmitter pairs that are connected to the processor with packet streams. The processor should get the process model created in 2b assigned. For the receivers/transmitters you have to set the supported packet format in the channel attribute to that type defined in 2a and set the data rate to 10000 (if you want you can choose another data rate, but you have to be sure to always use the same rate).

3. Create the models for the network nodes.
To be able to test the hub created above, you have to define network nodes, which are able to send and receive packets of our pre-defined format.

(a) Create a process model for the node.
The FSM of the node should look quite similar to that of the hub since here again the process waits for a packet and acts on it. The major difference is that packets can be received from two different entities: from the packet source, which is creating new packets to send, and from the point-to-point receiver.
In the first case the packet fields have to be filled and the packet has to be forwarded to the point-to-point transmitter.
When a packet has been received by the receiver, it simply has to be destroyed. For these two purposes two different transitions can be used. To distinguish between the two packet sources, e.g. the number of the stream

that caused the interrupt can be used (function op_intrpt_strm()).

In the function block you can enter a function doing all the stuff which is needed to send a packet. This includes getting a pointer to the packet, setting the fields correctly (for the 'id' you should create a state variable that is initialized in the init state and incremented each time a packet is sent) and sending the packet through the output stream.

The destination address should be randomly chosen. For that purpose you need a state variable pointing to a distribution, that is initialized with the command:
`address_dist = op_dist_load( "uniform_int", 0, n );`
with n being the number of nodes in the network minus one. To get an address for a packet, use the function op_dist_outcome().

The function of the transition for the packets received from another node should simply get a packet pointer and destroy the packet.

(b) Create a node model for the node.
The node model is again very simple. The following components are needed:

- A processor that uses the process model defined above.
- A point-to-point receiver and a transmitter.
- A processor that creates the packet (the packet source), which uses the simple_source process model to create packets of the format (!) defined in 2a.

Don't forget to set the correct packet format and data rate for transmitter and receiver. The packet interarrival time of the source should be set to a exponential distribution with mean outcome 10.

4. Create a link model.
A link model defines the properties of a link between two nodes. You should create a link model that supports the transmission of the self defined data packets with the apropriate data rate. To do so, select File/New and choose link model. The only supported link type should be 'ptdup', which indicates a point-to-point duplex connection. The supported packet formats field has to contain only the packet format from above and the data rate has to be set correctly. You also should select ecc_zero_err, no error model, dpt_propdel and dpt_txdel. Additionally you should select link_delay to be included as a external file, since this will be needed later. Don't forget to save everything.

5. Simulate a network containing the new hub and nodes.
Now you can use the above defined nodes to build a network and perform a simulation.

(a) Creating the statistics to collect some interesting data.
Now you have to define what data has to be collected. We want to measure the global delay in the network. For that purpose we measure the time each packet needs to reach its destination. This is done at the moment a node receives a packet.

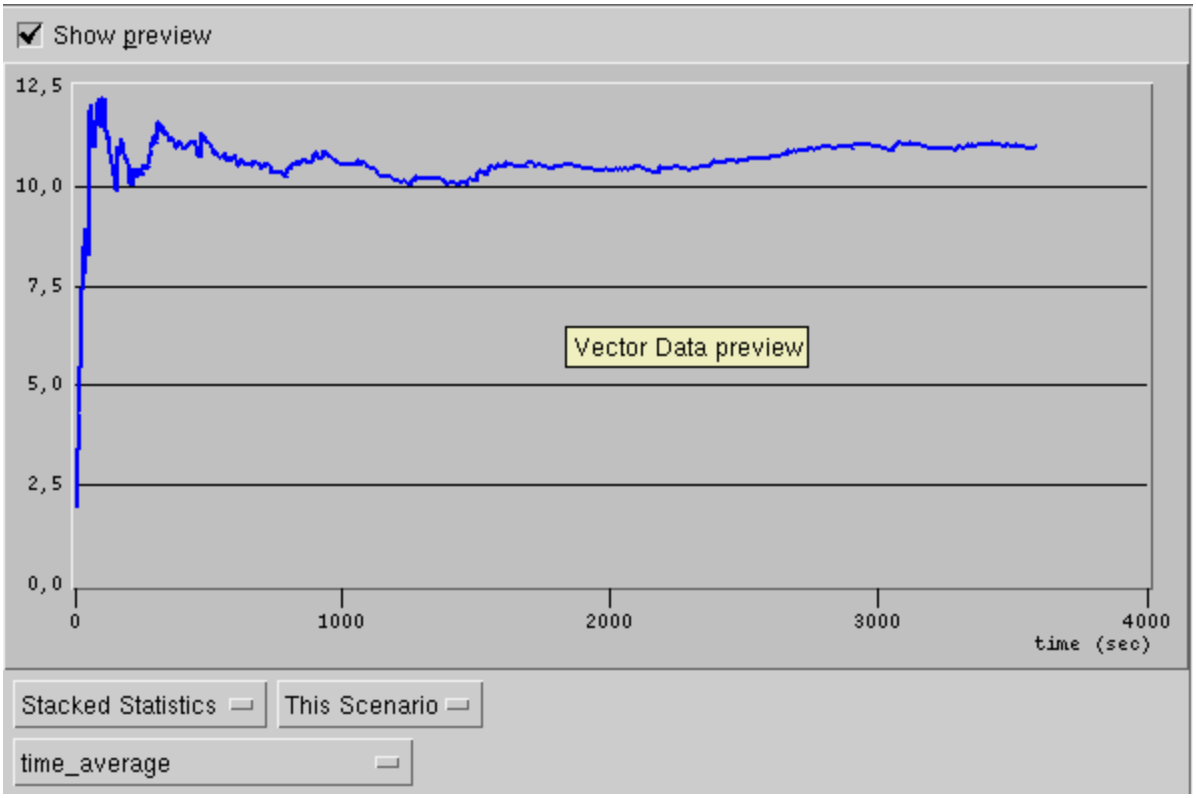
You should add a state variable of the type 'stathandle' to be able to access the statistic. In the init state you have to register the statistic (with op_stat_reg(...)). Inside the function of the correspondig transition you are able to compute the duration the received packet has needed to arrive at the current node (e.g. using op_sim_time() and op_pk_createion_time_get()) and write that value to the statistic.

Next, you have to add a global statistic to the process model of the node and name it apropriately.

(b) Building the network that has to be simulated.
The network should contain one hub with 8 nodes connected to it using the link model from above. You can use the 'Rapid configuration...' function to create it. To be able to select the self created models in the rapid configuration dialog, you have to customize the palette. Alternatively, you can create the network manually.

At the end you can check the connections with the 'verify links' function.

(c) Simulating the network and displaying the results.
To do the simulation, select the self created statistic with 'Choose individual DES statistics' and run the simulation. Open the resulting graph and interpret it. If you select 'time_average', the result should look similar to the following graphic:



List of useful functions:

- op_dist_load()
- op_dist_outcome()
- op_intrpt_strm()
- op_intrpt_type()
- op_pk_creation_time_get()
- op_pk_destroy()
- op_pk_get()
- op_pk_nfd_get_int32()
- op_pk_send()
- op_pk_nfd_set_int32()
- op_sim_time()
- op_stat_reg()
- op_stat_write()