Process Models

Steven Gordon

Sirindhorn International Institute of Technology
Thammasat University

June 2010
Contents

Example: Sensor Network

Process Modelling in OPNET

Controller Process Model

Simulation and Analysis
Custom Application for Sensor Network

Scenario

- Sensor network has sensors and 1 controller
- Sensors send updates to controller at regular frequency
- Controller may request a sensor to change update frequency
Custom Application for Sensor Network

Application runs directly on IP over IEEE 802.11 wireless LAN (no transport)
Project, Nodes and Processes

Project
SensorNetwork: 3 scenarios

Nodes
wireless_sensor_adv: sensing node
wireless_controller_adv: controller node

Processes
sensor: application on sensing node
controller: application on controller node
Contents

Example: Sensor Network

Process Modelling in OPNET

Controller Process Model

Simulation and Analysis
State Transition Diagrams: States

Actions performed by code in **enter executives** and **exit executives** of states

**Unforced states:**
- Upon invocation, exit executives run in current state then enter executives in next state
- Blocked (waits for interrupt) between enter and exit executives
- Models true states

**Forced states:**
- No blocking: enter/exit executives run during invocation
State Transition Diagrams: Transitions

Transitions may contain: condition/executive

- If condition true, enter next state
- Run executive code if exists

Designer must ensure 1 and only 1 condition can be true at a time

Actions are combination of code in: exit executive, transition executive, enter executive
Variables

Variables can be scoped as:

**Temporary**  locally available in executives only

**Function**  locally available in functions only

**State**  global across executives and functions

**Global**  global across executives, functions and external code

Data types (other than C data types):

- Evhandle, Ici, Objid, Packet, Compcode, List, Statethandle, Distribution
Source Code

Executives  state and transition executives
Header Block includes, defines and declarations for process model
Function Block  functions used by process model
Diagnostic Block  debugging code
Termination Block  executed when simulation finishes
External Code  headers/functions stored externally from process model
Contents

Example: Sensor Network

Process Modelling in OPNET

Controller Process Model

Simulation and Analysis
Controller Process Model
Controller Process Model

Four states:

- **INIT** performs initialization operations for process model
- **Idle** process resides here while waiting for interrupts
- **Receive** entered when the packet is received from a lower layer
- **Send** entered when a packet needs to be sent to the lower layer

Transitions:

- **PKT_RECEIVE** stream interrupt occurs due to packet arriving on input stream from `ip_encap`
- **PKT_SEND** self interrupt occurs due to timeout
Initialising the Process

INIT state:
1.Initialises state variables
2. Registers itself with IP
3. Schedules self interrupt to occur after a random time

controller_sv_init()
1. Obtains ID of itself (controller process) and parent (wireless_controller_adv node)
2. Reads user-configurable process attributes
3. Registers statistics
4. Create ICI to communicate with IP
5. Establishes a random distribution to choose address of sensor
Initialising the Process

**Task**
Add code to read the *Interarrival Time* attribute

**User-configurable Process Attributes**

*Interfaces → Model Attributes*: attributes that can be set in the node model (or promoted to higher level)

Programmers Reference → Discrete Event Simulation → Internal Model Access
Models → General Models → Model Support
std/utilities/oms/oms_dist_support.ex.c
Initialising the Process

```python
controller_register_self_with_ip()
```

- `ip_encap` performs multiplexing: multiple higher layer protocols over IP
- Must register our higher layer protocol with IP

After initialisation is complete, controller enters *Idle* state and waits for next event
Events from Idle State

Receive a packet: stream interrupt from lower layer

```c
#define PKT_RECEIVE op_intrpt_type () == OPC_INTRPT_STRM
```

Send a packet: self interrupt (timeout)

```c
#define PKT_SEND op_intrpt_type () == OPC_INTRPT_SELF
```

Interrupts in general:

- Type: stream (packet), self (timeout), stat (status), ...
- Code: user-programmed, e.g. indicate which timer expired
- For stream interrupts, can determine which stream, e.g. from higher layer or lower layer
- Check the interrupt in exit executives of unforced states

Programmers Reference → Discrete Event Simulation → Interrupt Package
Receiving a Packet

Process the packet:

1. Get the received packet from the stream
2. Get the ICI that accompanies the packet
3. Update the statistics
4. Destroy the packet and ICI

Return to *Idle* state
Sending a Packet

Generate the packet:

1. Create random sized packet with random update frequency and destination
2. Send the packet (controller_packet_send_to_ip())
3. Update the statistics
4. Schedule self interrupt for next request for update frequency change

controller_packet_send_to_ip()

1. ICI contains address of destination sensor
2. Packet is sent

Programmers Reference→Discrete Event Simulation→Interface Control Information Package
Programmers Reference→Discrete Event Simulation→Packet Package
Sensor Process Model

Similar to controller process model: must regularly send packets, also send immediately after receiving packet

Task
Add code to retrieve the update period from the received packet

Programmers Reference ➔ Discrete Event Simulation ➔ Event Package
Contents

Example: Sensor Network

Process Modelling in OPNET

Controller Process Model

Simulation and Analysis
**Simulation and Analysis**

**Tasks**

1. Run the simulation
2. Show the existing result templates: DES $\rightarrow$ Panel Operations $\rightarrow$ Arrange Panels $\rightarrow$ Show All
3. Load the results into the templates: DES $\rightarrow$ Panel Operations $\rightarrow$ Panel Templates $\rightarrow$ Load with Latest Results

Results show the regular updates from sensors, and change in update interval triggered by controller
Statistics, Buckets and Sum/Time

- Buckets are used to reduce the statistic data stored during simulation, e.g. store 1 value for all packets sent in bucket duration of 10ms
- Sum/Time is often used for rate statistics, e.g. bits per second
- Last value in a bucket is proportionally split across current bucket and next bucket
- To avoid this with sum/time statistics, write value 0 after writing real statistic
- Default bucket size: sim duration / values per statistic