OPNET Modeler Overview

NOTE: This material is for in-class use only.
Do not replicate or distribute.

Event-Driven Simulation

- An event is a request for a particular activity to occur at a certain time.
- OPNET simulations are event-driven. Time, in the simulation, advances when an event occurs.
- A different method might be to sample at regular intervals. Disadvantages are as follows:
  - Accuracy of results is limited by the sampling resolution.
  - Simulation is inefficient if nothing happens for long periods.

Event List Concepts

- An OPNET simulation maintains a single global event list.
- All objects access a shared simulation time clock.
- Events are scheduled on the list in time order. The first event on the list is the head.

Head

Time	Event Type	Module
0.0	Initialize	src.gen
0.0	Initialize	src.rte
4.3	Timer expires	src.gen
4.3	Packet arrives	src.rte

- An event has data associated with it.
- When an event completes it is removed from the list.

Interrupts

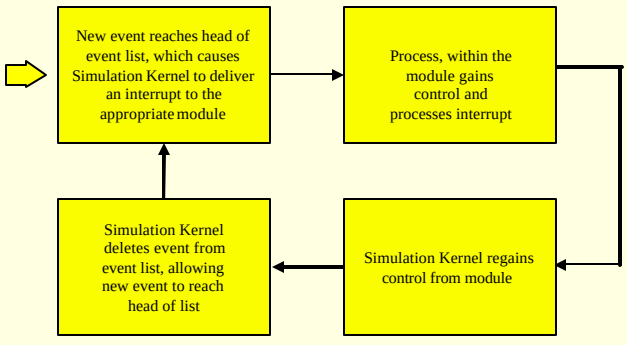
- An event becomes an interrupt when it reaches the head of the event list and is delivered by the Simulation Kernel to the designated module.
- Data associated with the event can be obtained by the module when the interrupt occurs.
- Certain modules, processes, and queues can be selected to place initial interrupts on the event list.

The Simulation Kernel

- An entity, the Simulation Kernel (SK), manages the event list.
- The SK delivers each event, in sequence, to the appropriate module.
- The SK receives requests from modules and inserts new events on the event list.

5

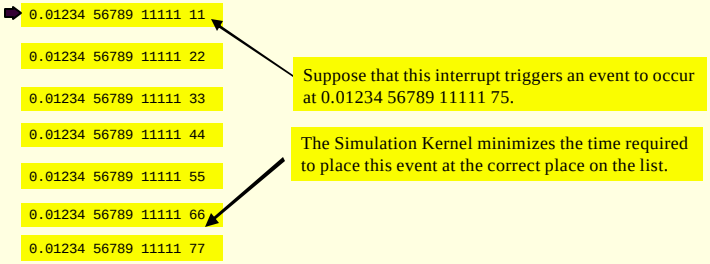
How Does the Event List Work?



6

Event List Implementation

- The Simulation Kernel uses a proprietary, efficient algorithm to maintain the event list.
- Event times are expressed as double-precision, floating point numbers and are used to keep the event list sorted.



Simultaneous Events

- What Happens When Two Events Are Scheduled for the Same Simulation Time?
- The events are actually simulated sequentially, though they appear to occur synchronously according to the time clock.
- The Simulation Kernel uses one of two methods to determine execution order:
  - "Natural order method": The event that reaches the list first is executed first.
  - "Priority factor method":
    - Modules and events are assigned priority factors.
    - Events with a higher priority or originating from a higher priority module are executed before events with a lower priority or from a lower priority module.

Event List Concepts Reviewed

- Some events must be entered on the event list at the start of a simulation.
  - A generator module enters an initial event.
  - A processor or queue module has the `begsim interrupt` attribute enabled.
- An event list typically has a few events – each event spawns another event or two that is placed on the list as the spawning event is deleted.
- The event list is always growing and shrinking.
- An event is pending until executed. A pending event can be cancelled.

Delivery of Interrupts

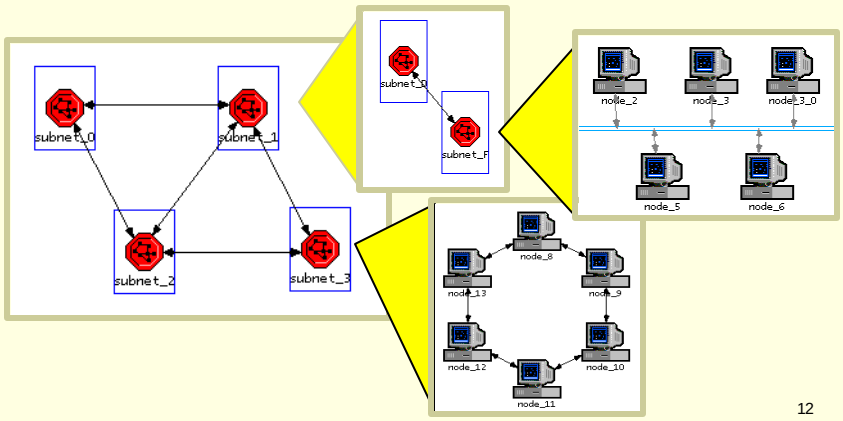
- When an interrupt is delivered to a module, control passes from the Simulation Kernel to the module.
- If the module is a queue or processor, the interrupt is delivered to the process running within the module.

Network and Node Models

Network Objects - Subnets

- A subnetwork abstracts network components specified within it into a single object.
- Subnetworks represent identical constructs in an actual network.

Subnetwork



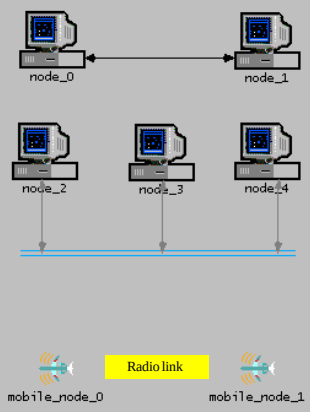
Network Objects - Links

- Link objects model physical layer effects between nodes, such as delays, noise, etc.

A point-to-point link transfers data between two fixed nodes.

A bus link transfers data among many nodes and is a shared media.

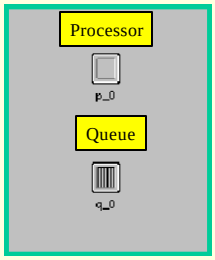
A radio link, established during a simulation, can be created between any radio transmitter-receiver channel pair. Satellite and mobile nodes must use radio links. Fixed nodes may use radio links. A radio link is not drawn but is established if nodes contain radio transceivers.



13

Node Objects - Modules

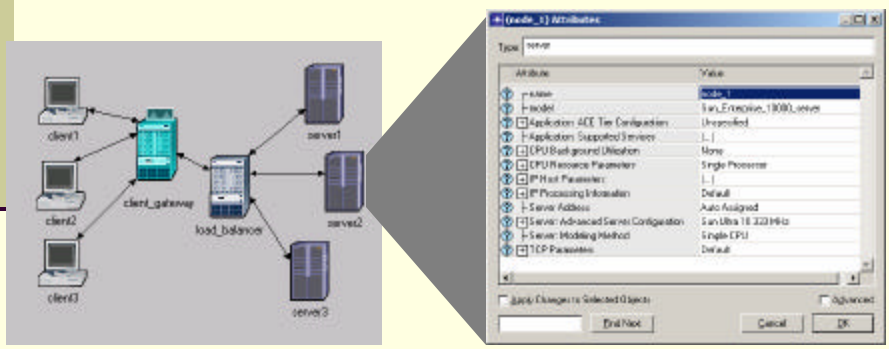
- Modules are the basic building blocks of node models. Modules include processors, queues, transceivers, and generators.
- Processors are the primary general purpose building blocks of node models, and are fully programmable.
- Queues offer all the functionality of processors, and can also buffer and manage a collection of data packets.



14

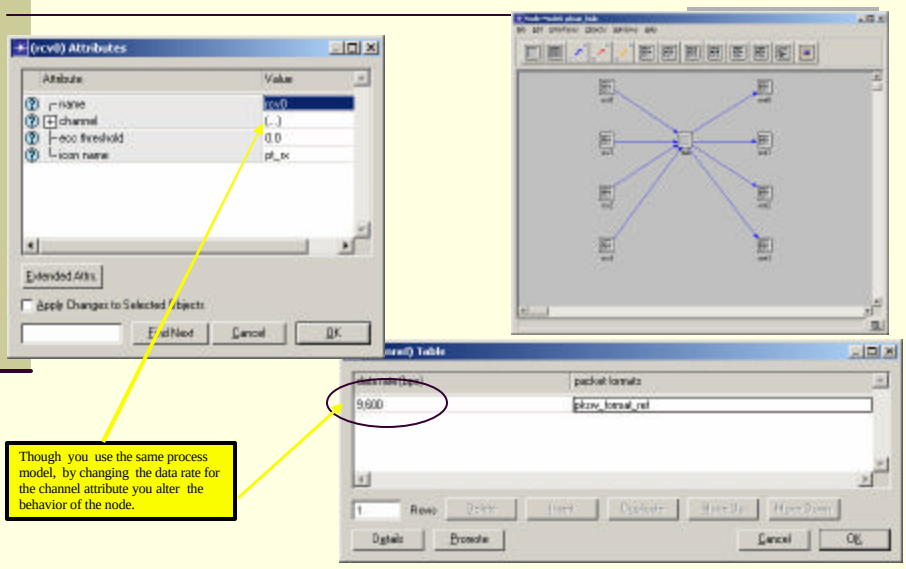
Object Attributes

- Attributes are parameters of an object that can configure its behavior.
- Attributes are dynamically changeable during simulation.
- Processes have access to all object attributes.
- Different attribute values allow objects of the same type to behave differently.



15

Object Attributes



Assigning Attribute Values

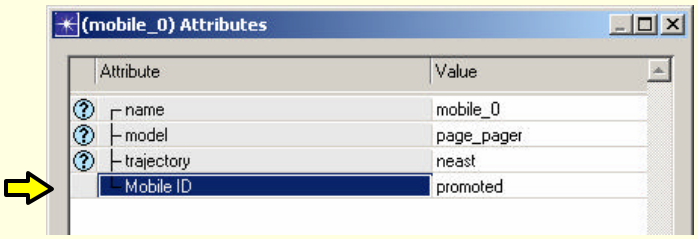
- You can assign attribute values by right-clicking on an object and selecting or specifying the attribute value.
- Attributes are of a certain type. Commonly used types are listed.

Type	Definition
Integer	Whole numbers: storage capacities; transmission window size
Double	Decimal numbers: processing speeds; timer values
String	General text info: statistic names, object names, options
Toggle	True/false condition: status flags, semaphores
Typed file	User defined file: routing tables, address mappings, script file
Compound	Nested, complex data: routing table, circuit table, subqueues

17

Promoting Attribute Values

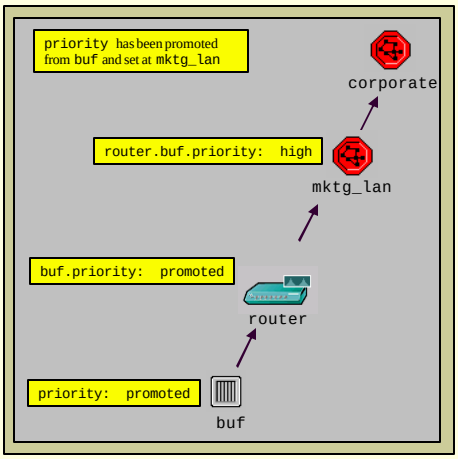
- You can "promote" an attribute. This means that you assign a value at a higher hierarchical level.
- Passing control of a lower-level object to a higher level provides more flexibility in how objects are used.
- You can leave an attribute unspecified at even the network level, and assign a value at run time.



18

Promoting Attributes Example

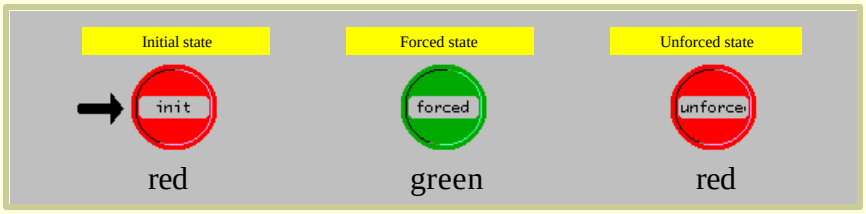
- When an attribute assignment is made, promotion stops. An attribute value was assigned at `mktg_lan`, so the attribute does not appear in the object `corporate`.
- Attribute names are used as prefixes at each new level of the object hierarchy.



Process Model

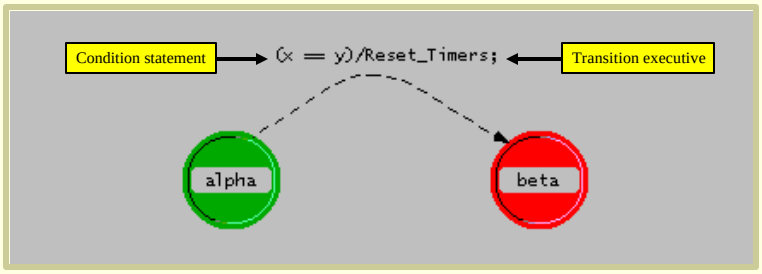
Process Model Objects - States

- The *initial state* is the place where execution begins in a process.
- A *forced state* does not allow a pause during the process.
- An *unforced state* allows a pause during the process.
- Later chapters will fully discuss the differences between these types of states.



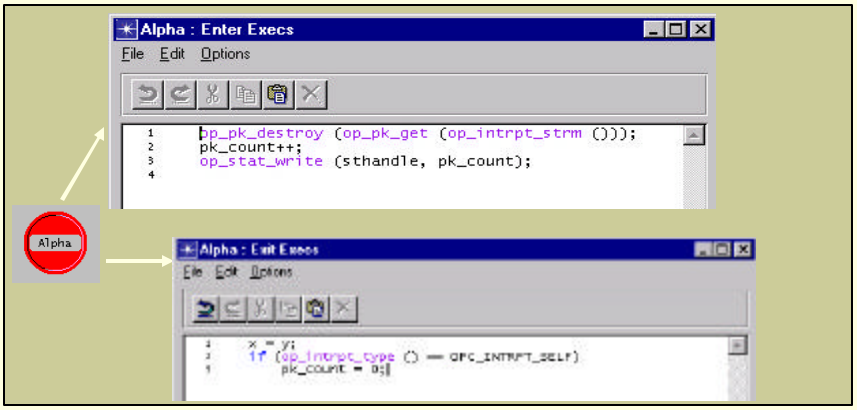
State Connections - Transitions

- Transitions describe the possible movement of a process from state to state and the conditions allowing such a change.
- Exactly one condition must evaluate to true.
- If the condition statement *(x == y)* is true, the transition executive *(Reset_Timers;)* is invoked.



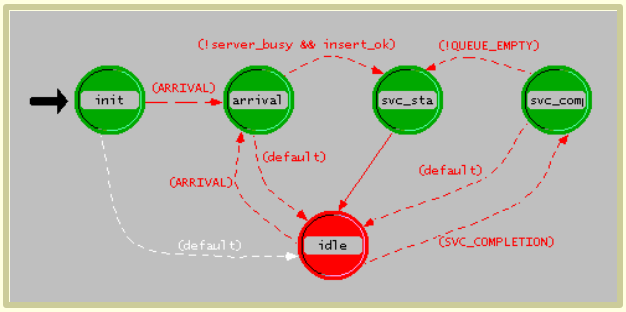
Executive blocks

- Each state has two executive blocks
  - *Enter executives* are invoked on entering a state.
  - *Exit executives* are invoked before exiting a state.



What is Proto-C™ ?

- Proto-C consists of
  - State transition diagrams
  - The complete C programming language
  - The library of OPNET Kernel Procedures (KPs) State variables (private to each process)
  - Temporary variables



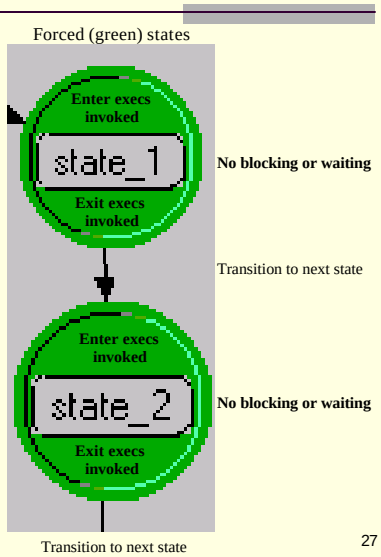
Process Model

Process Models

- A process in the context of computer systems and communications networks can be viewed as a series of logical operations performed on data, and the conditions that cause these operations.
- Processes may be implemented in terms of both hardware or software components.
- OPNET *process models* describe the logic of real-world processes, such as:
  - Communications protocols and algorithms
  - Shared resource managers
  - Queuing disciplines
  - Specialized traffic generators
  - Statistic collection mechanisms
  - Operating systems
- The Process Editor provides the necessary features for specifying process models, which consist of both graphical and textual components.

Forced States

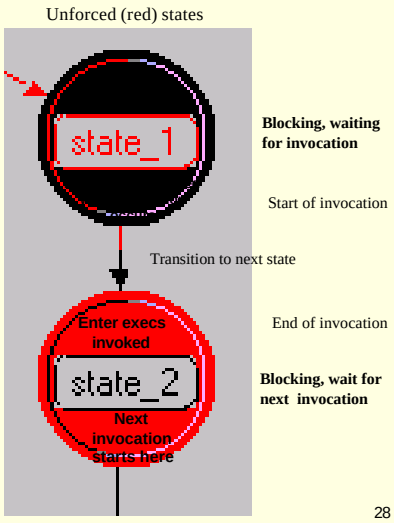
- Forced (green) and unforced (red) states differ significantly in execution timing.
- In a forced state, the process:
  - Invokes the enter executives
  - Invokes the exit executives
  - Evaluates all condition statements
  - If exactly one condition statement evaluates to true, the transition is traversed to the next state.



27

Unforced States

- In an unforced state, the process
  - Invokes the enter executives
  - Places a marker at the middle of the state
  - Releases control to the Simulation Kernel and becomes idle
  - Resumes at the marker and processes the exit execs when next invoked



28

Transitions Between States

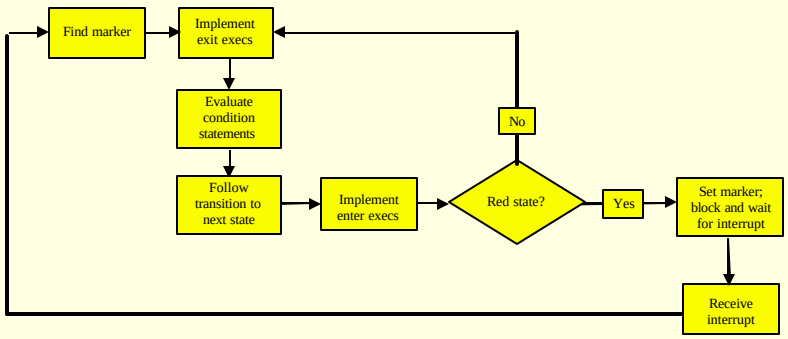
- After completing the exit executives, the process evaluates the condition statements of all departing transitions from the state.
- One and only one condition statement must evaluate to true.
- The process traverses the transition associated with this condition statement.
- A transition with condition = “default” is true if and only if no other conditions are true.
- A transition with no condition set is termed *unconditional* and is always true.

29

How a Process Handles an Interrupt

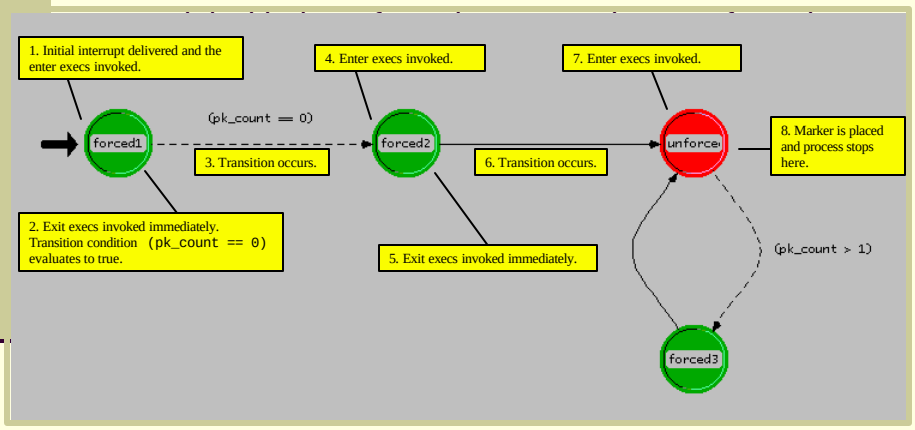
- Flow diagram showing how a process handles an interrupt

(except the initial interrupt)



30

Process Model Example

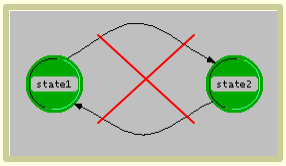


Simulation Termination

- Simulations terminate in one of four ways
  - The event list is emptied.
  - Simulation attribute `duration` expires.
  - A process calls for termination, using the KP ***op_sim_end().***
  - A fatal error occurs.

How Does Time Advance?

- Simulation time advances only when an event with a later time is taken from the event list.
- No simulation time occurs during an invocation of a process model.
- No time elapses during transitions between states.
  - A process model must always end in a red state so time can advance.
  - Avoid endless looping between forced (green) states.



33

Node Model

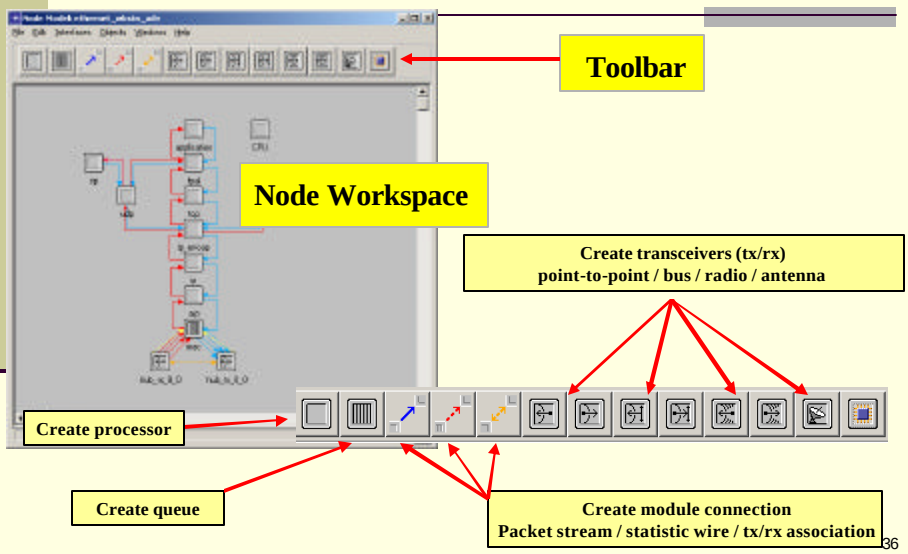
34

Node Editor

- The Node Editor provides the resources necessary to model the internal functions of nodes.
- Users have access to different modules which are used to model internal aspects of node behavior.
- Modules represent the internal capabilities of a node such as:
  - •Data creation
  - •Transmission
  - •Reception
  - •Storage
  - •Internal routing
  - •Queuing

35

Node Editor



36

Node Editor - Toolbar

Processor. A module that represents the most general building block of node models. The behavior of a processor can be completely specified by the user and its links can be arbitrarily connected to other modules.

Queue module. A module that provides a superset of the functionality of processor modules. Queue modules can execute an arbitrary process model that describes the behavior of a particular process or protocol, and can be connected via packet streams to other modules.

37

Node Editor - Toolbar

Packet stream. A connection between modules that carries data packets from a source module to a destination module. They represent the flow of data across the hardware and software interfaces within a communications node

Statistic wire. A connection between modules that conveys numeric values between devices or processes in the same node. Statistic wires are primarily used to allow processes to monitor changes in state and performance of the devices that make up a node, and to create a simple signaling mechanism between processes.

Logical association. A connection used to indicate that a relationship exists between two modules in a node model, for example, between a receiver and transmitter used as a pair. Logical associations do not carry any data.

38

Node Editor - Toolbar

Transmitters: the outbound interface between packet streams inside a node and communications links outside the node.


Point-to-point


Bus


Radio

Receivers: the inbound interface between communications links outside a node and packet streams inside a node.


Point-to-point


Bus


Radio

Antenna: A module that is used to specify the antenna properties for radio transmitter or receiver modules.

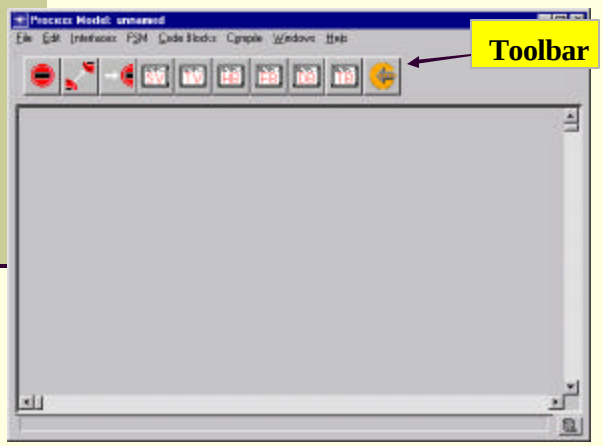

Antenna

39

Process Model

40

Process Model Editor



1. Create state
2. Create transition
3. Set initial state
4. Edit state variable
5. Edit temporary variable
6. Edit header block
7. Edit function block
8. Edit diagnostic block
9. Edit termination block
10. Compile process model

41

Process Editor Toolbar

Create State: Creates a new state within the process model.

State: One of the components of a finite state machine. The behavior of a state is defined by its state executives, which are executed upon entry into and exit from the state.

Create Transition: Creates transitions between states.

Transition: The path a process takes between states in a finite state machine. Transitions contain attributes that can be used to specify conditions that must be met before the transition takes place.

Set Initial State: Sets the selected state in the process to be the initial one.

42

Process Editor Toolbar

State Variables Block: Defines variables that retain their value from one process invocation to the next.

Temporary Variables Block: Defines variables that retain their value only during the span of a single process invocation.

Header Block: Defines constants, macro expressions, include files, global variables, data structures, data types, and function declarations for the process. Also declares whether the process model will be in C/C++

Function Block: Defines C/C++ functions that are associated with the process.

43

Process Editor Toolbar

Diagnostic Block: Defines C/C++ statements that send diagnostic information to the standard output device.

Termination Block: Defines C/C++ statements that execute just before a process is destroyed.

Compile Code: Generates the C/C++ source file and object code for the process model.

44

Creating Process Models

Follow these steps

1. Understand the questions to be answered.
2. Create a new process model or modify an existing process model.
3. Edit the node model to use the new/modified process model.
4. Modify the existing probe file.
5. Specify the simulation sequence file.
6. Determine the expected output.
7. Run simulations.
8. Analyze raw output and post-process it to answer questions.
9. Compare actual results to expected output. Explain any differences.

45

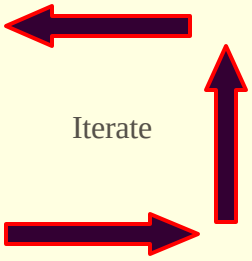
OPNET : Projects and Scenarios

- Modeler uses a Project-and-Scenario approach to modeling networks.
- A *Project* is a collection of related network scenarios in which each explores a different aspect of network design. All projects contain at least one scenario.
- A *Scenario* is a single instance of a network. Typically, a scenario presents a unique configuration for the network, where configuration can refer to aspects such as topology, protocols, applications, baseline traffic, and simulation settings.

46

The Project/Scenario Workflow

- Create project
- Create baseline scenario
  - Import or create topology
  - Import or create traffic
  - Choose results and reports to be collected
  - Run simulation
  - View results
- Duplicate scenario
  - Make changes
  - Re-run simulation
  - Compare results



47