Introduction to OMNeT++

Acknowledgment

- The source material for this presentation was borrowed from the “OMNeT++ User Manual Version 4.1”

What is OMNeT++

- OMNeT++ is an object-oriented modular discrete event simulation framework
- It is not a simulator of anything concrete, but rather provides infrastructure and tools for writing simulations
  - Specific problem domains are modeled using specialized component libraries built on top of the OMNeT++ simulation framework
- Models are assembled from reusable components termed modules and can be combined in various ways like LEGOs
- OMNeT++ is highly portable and runs on most common operating systems (e.g., Linux, OS X, Windows)

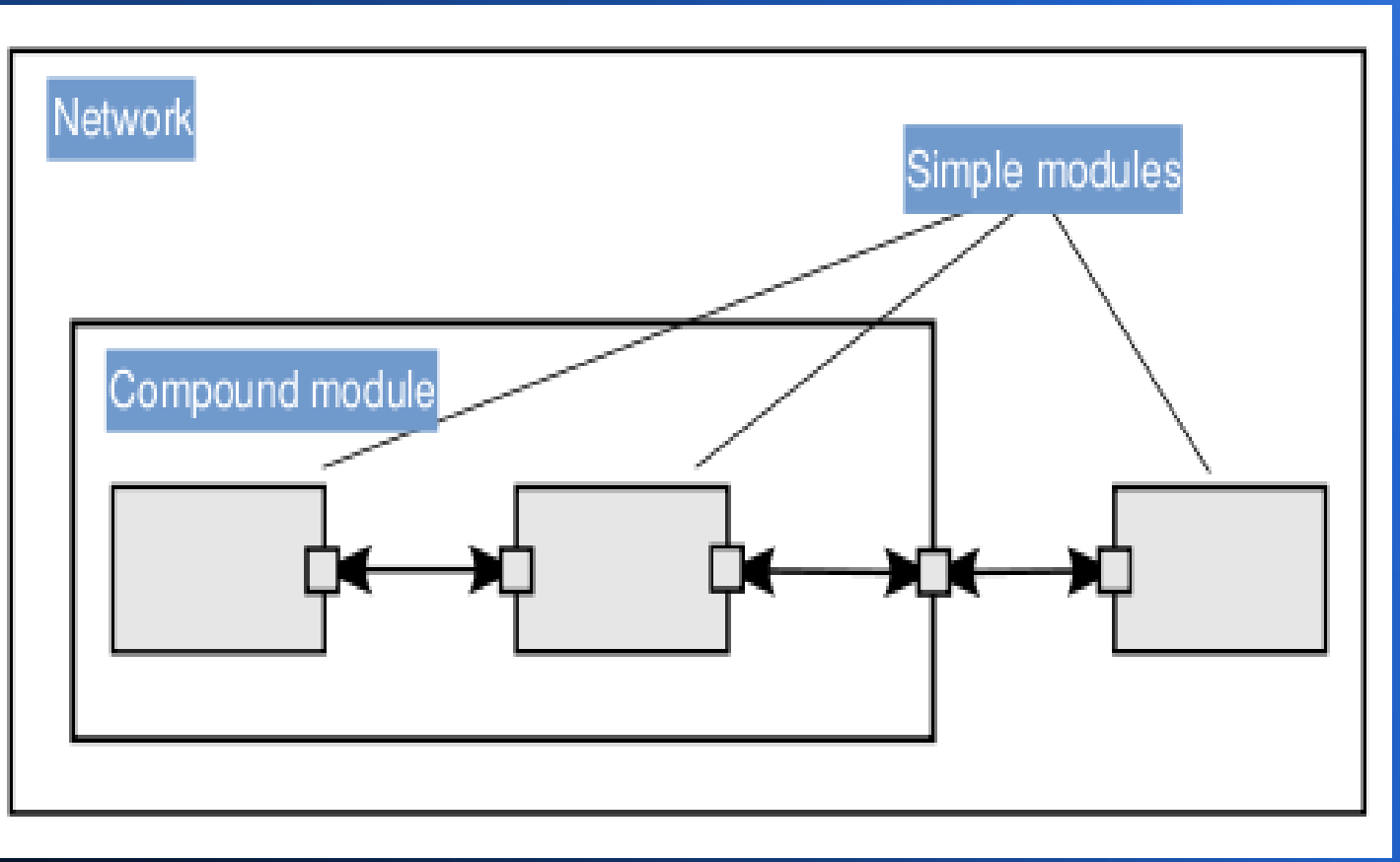
Example Problem Domains

- Modeling of wired and wireless communication networks
- Protocol modeling
- Modeling of queueing networks
- Modeling of multiprocessors and other distributed hardware systems
- Validating of hardware architectures
- Evaluating performance aspects of complex software systems

Hierarchical Modules

- An OMNeT++ model consists of hierarchically nested modules that communicate by passing messages to each other
- The top level module is called the *system module* which contains submodules that can also contain submodules themselves
- Depth of module nesting is unlimited, allowing the user to reflect the logical structure of the actual system in the model structure
- The active modules are termed *simple modules* and are written in C++ using the simulation class library
- Simple modules can be grouped into *compound modules* and so forth; the number of hierarchies is unlimited
- The whole model, called a *network*, is itself a compound module

Modeling Concepts



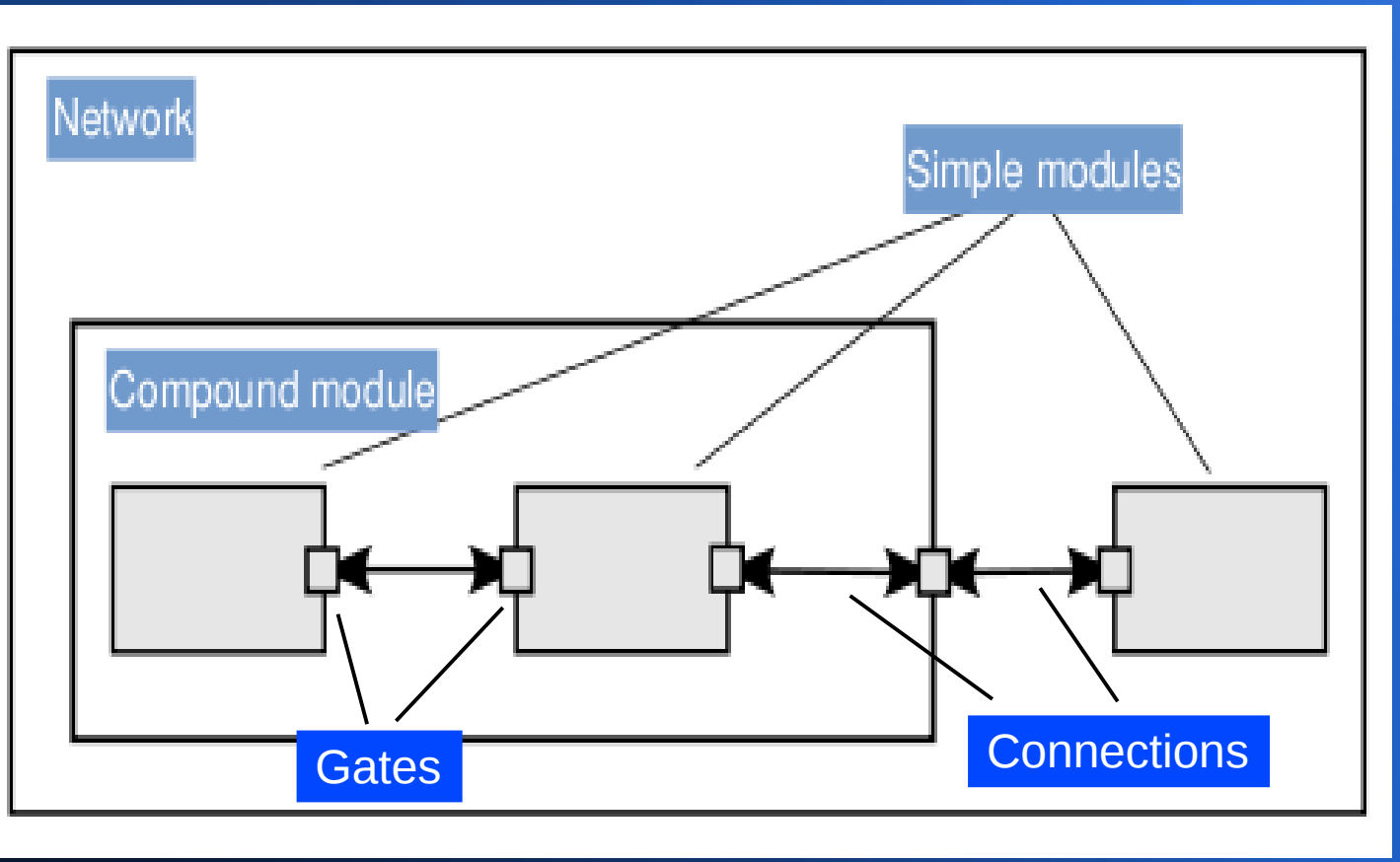
Module Types

- Simple and compound modules are instances of *module types*
- In describing the model, the user defines module types; instances of these module types serve as components for more complex module types
- Finally, the user creates the system module as an instance of a previously defined module type; all modules of the network are instantiated as submodules and sub-submodules of the system module
- Module types can be stored in files separately from the actual place of usage allowing for grouping of existing module types to create *component libraries*

Messages, Gates, Links

- Modules communicate by exchanging messages
- The local simulation time of a module advances when the module receives a message – the message can arrive from another module or from the same module (self-messages are used for timers)
- Gates are the input and output interfaces of modules; messages are sent through output gates and arrive through input gates
- Each connection is created within a single level of the module hierarchy
- Because of the hierarchical structure of the model, messages typically travel through a series of connections, starting and arriving in simple modules

Modeling Concepts



Modeling of Packet Transmissions

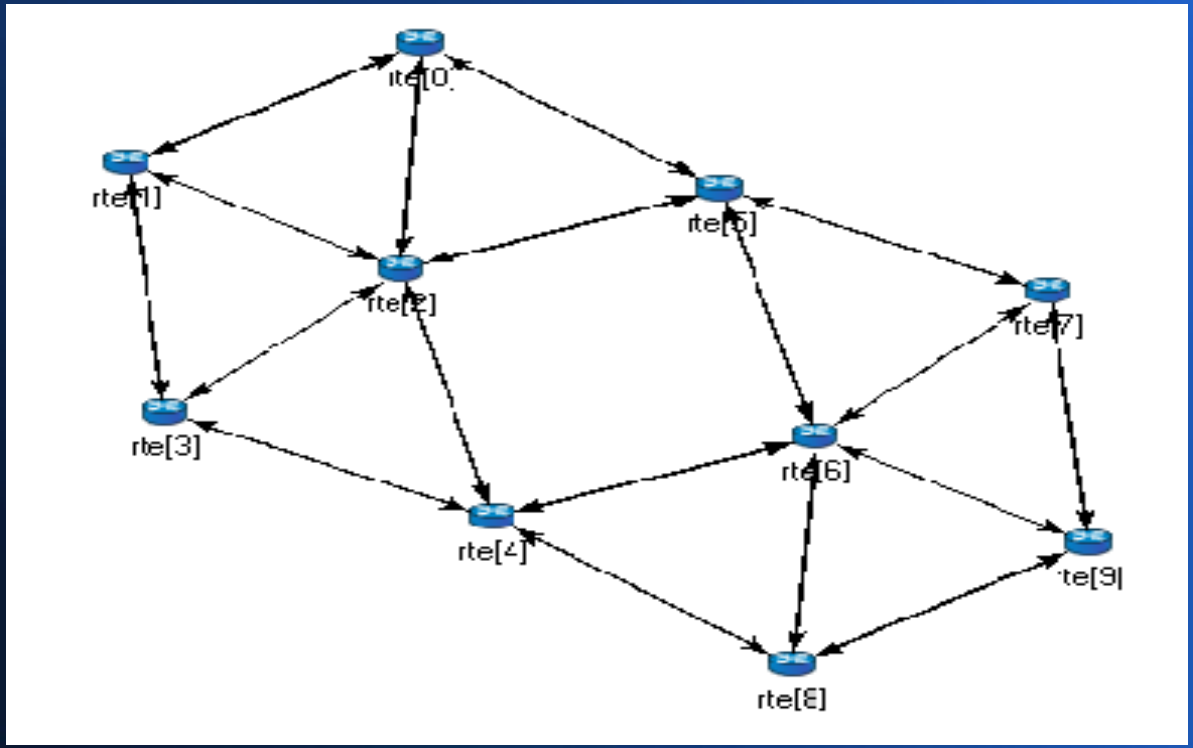
- To facilitate the modeling of communication networks, connections can be used to model physical links
- Connections support the following parameters: data rate, propagation delay, bit error rate and packet error rate, and may be disabled
- These parameters and the underlying algorithms are encapsulated into *channel* objects
- The user can parameterize the channel types provided by OMNeT++ and also create new ones

Parameters

- Modules can have parameters which can be assigned in either the NED files or the configuration file omnetpp.ini
- Parameters can be used to customize simple module behavior and to parameterize the model topology
- Parameters can take string, numeric or boolean values, or contain XML data trees
- Within a compound module, parameters can define the number of submodules, number of gates and the way the internal connections are made

Topology Description Method

- The user defines the structure of the model in the NED language descriptions (Network Descriptions) - identifies the network's nodes and the links between them



Programming the Algorithms

- The simple modules contain algorithms as C++ functions
- The full flexibility and power of C++ can be used, supported by the OMNeT++ simulation class library
- The simulation programmer can choose between event-driven and process-style description, and freely use object-oriented concepts (inheritance, polymorphism, etc.) and design patterns to extend the functionality of the simulator
- Simulation objects are represented by C++ classes

Simulation Class Library

- The simulation class library includes:
  - Module, gate, parameter, channel
  - Message, packet
  - Container classes (e.g., queue, array)
  - Data collection classes
  - Statistics and distribution estimation classes (histograms, P^2 algorithms for calculating quantities, etc.)
  - Transient detection and result accuracy detection classes
- The classes have been designed to work together efficiently, creating a powerful simulation programming framework

Building and Running Simulations

- OMNeT++ model consists of following parts:
  - NED language topology description (.ned files)
  - Message definitions (.msg files)
  - Simple module sources (C++ files with .h/.cc suffixes)
- The simulation system provides the following components:
  - Simulation kernel – code that manages the simulation and the simulation class library
  - User interfaces – used in simulation execution to facilitate debugging, demonstration, or batch execution of simulations
- Simulation programs are built from the above components and may be compiled as a standalone program executable or as a shared library

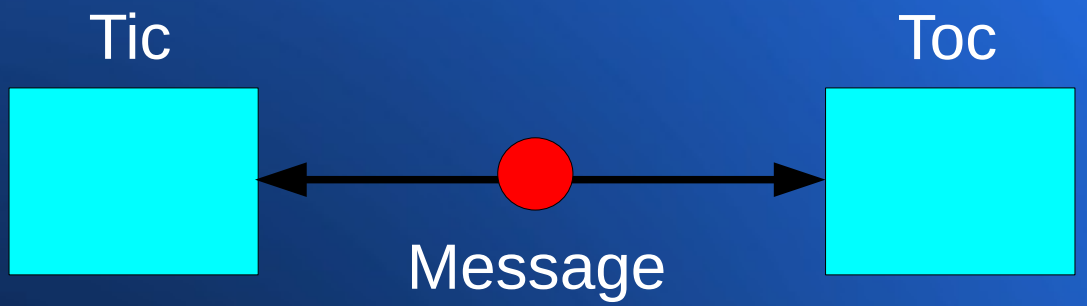
Analyzing the Results

- The output of the simulation is written into result files: output vector files, output scalar files and the user's own output files
- OMNeT++ contains an IDE that provides a rich environment for analyzing these files
- Output files are line-oriented text files which can be processed by a variety of tools such as Matlab, GNU R, Perl, Python and spreadsheets

Specialized Component Libraries

- Partial list of OMNeT++-based network simulators and simulation frameworks:
  - Mobility Framework -- for mobile and wireless simulations
  - INET Framework -- for wired and wireless TCP/IP based simulations
  - Castalia -- for wireless sensor networks
  - MiXiM -- for mobile and wireless simulations
  - OverSim -- for overlay and peer-to-peer networks (INET-based)
  - NesCT -- for TinyOS simulations
  - Consensus Positif and MAC Simulator -- for sensor networks
  - SimSANs -- for storage area networks
  - CDNSim -- for content distribution networks

Simulation Example



- We will simulate a network that consists of two nodes
- The nodes will do something simple: one of the nodes will create a packet, and the two nodes will keep passing the same packet back and forth
- The nodes will be called "tic" and "toc"

Questions?