Présentation d'OMNET++

Jeux de transparents tiré de

Introduction to simulation with OMNET++

José Daniel García Sánchez – ARCOS Group – University Carlos III of Madrid

NOTES: Utilisé à des fins d'enseignements, certains transparents ont été retirés.

OMNeT++ Discrete Event Simulation System

INTRODUCTION TO SIMULATION WITH **OMNET++**

José Daniel García Sánchez
ARCOS Group – University Carlos III of Madrid

What is OMNET++?

- A discrete event simulation environment.
  - Mainly focused in communication networks.
  - Runs on Linux and Windows.
  - C++ based.
  - Free for academic and non-profit use.
  - GUI support.
  - Supports parallel execution (MPI based).
  - Several component add-on libraries available.

Available off-the-shelf models

- TCP/IP.
  - **IP.**
  - **TPC.**
  - **UDP**
  - **PPP.**
  - …
- Network protocols
  - **Ethernet.**
  - **802.11.**
  - FDDI.
  - Token Ring.
- Peer-to-peer.
- Sensor networks.

Important issues in a discrete event simulation environment

- Pseudorandom generators.
- Flexibility.
- Programming model.
- Model management.
- Support for hierarchical models.
- Debugging and tracing.
- Documentation.
- Large scale simulation.
- Parallel simulation.
- Experiment specification.

Randomness

- Several random number generators
  - Including Mersenne-Twister.
  - Easy to plug-in custom generators.
- Mechanism for generating most random variates.
  - 14 continuous distributions.
  - 6 discrete distributions.
  - Very easy to define your own distributions.
  - Capability to generate a distribution from a sample.

Flexibility

- Core framework for discrete event simulation.
  - Different add-on for specific purposes.
- Fully implemented in C++.
- Functionality added by deriving classes following specified rules.

Programming model

- Key elements:
  - Topology: Describes relationship among elements.
  - Behavior: Describes how a node behaves.

Topology

- NED Language
  - Describes links among nodes.
  - Describes composition model.
  - May be text edited or GUI.
- Possibility to create topology at run-time.

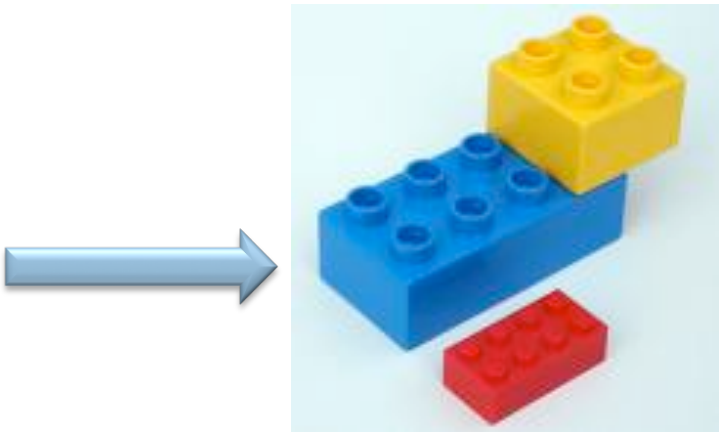
Behavior

- C++ Code automatically generated.
- Only methods describing behavior are needed to be redefined.

Model management

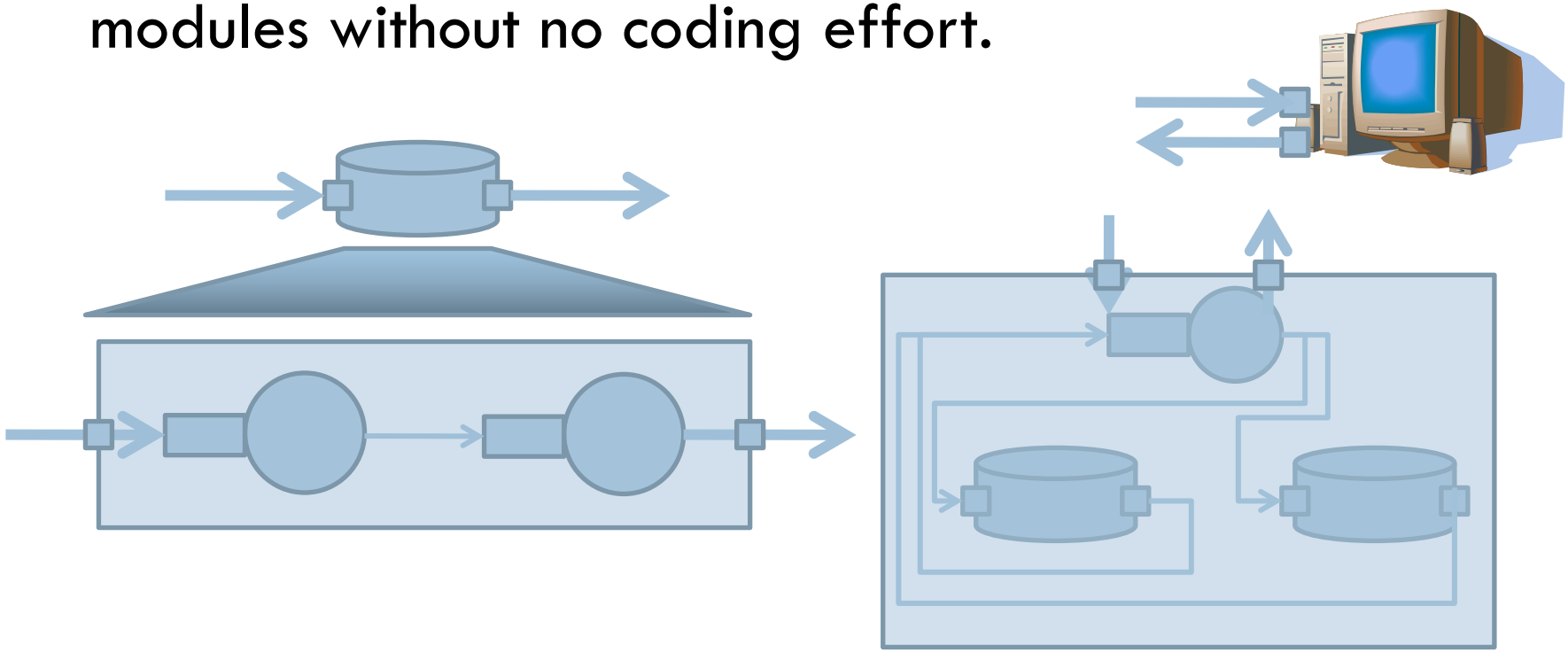
- Clear separation among simulation kernel and developed models.
- Easiness of packaging developed modules for reuse.
- No need for patching the simulation kernel to install a model.

Build models and combine like
LEGO blocks



Hierarchical modeling

- Modules are self-contained and reusable.
- Compound models may be assembled from simpler modules without no coding effort.



Debugging and tracking

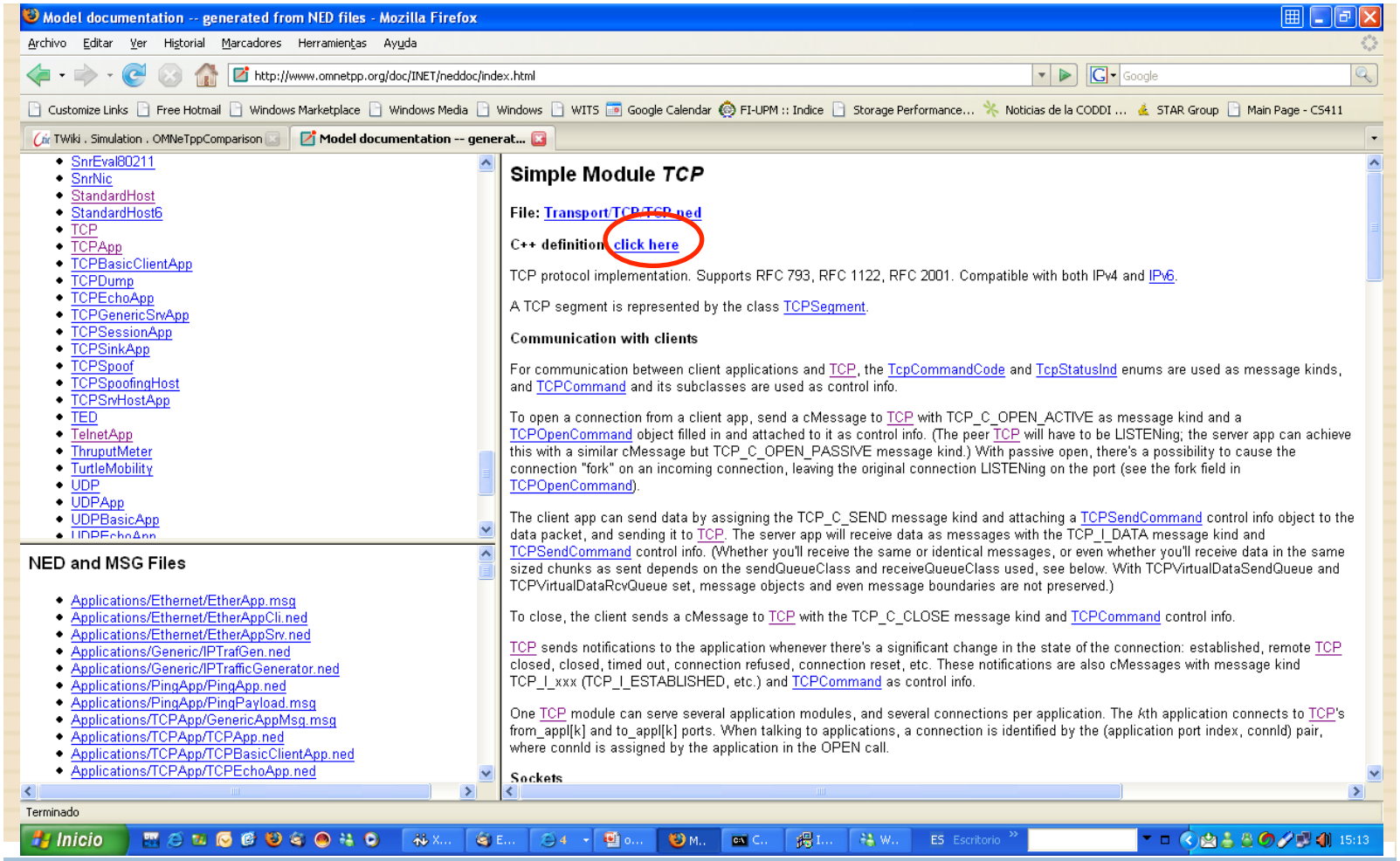
- Simulations may be run in two modes:
  - Command line: Minimum I/O, high performance.
  - Interactive GUI: Tcl/Tk windowing, allows view what's happening and modify parameters at run-time.

- Additional support for tracing.

Documentation

- Well structured documentation.
  - Well written and complete user's manual. **247 pages**.
  - Well documented (doxygen) API.

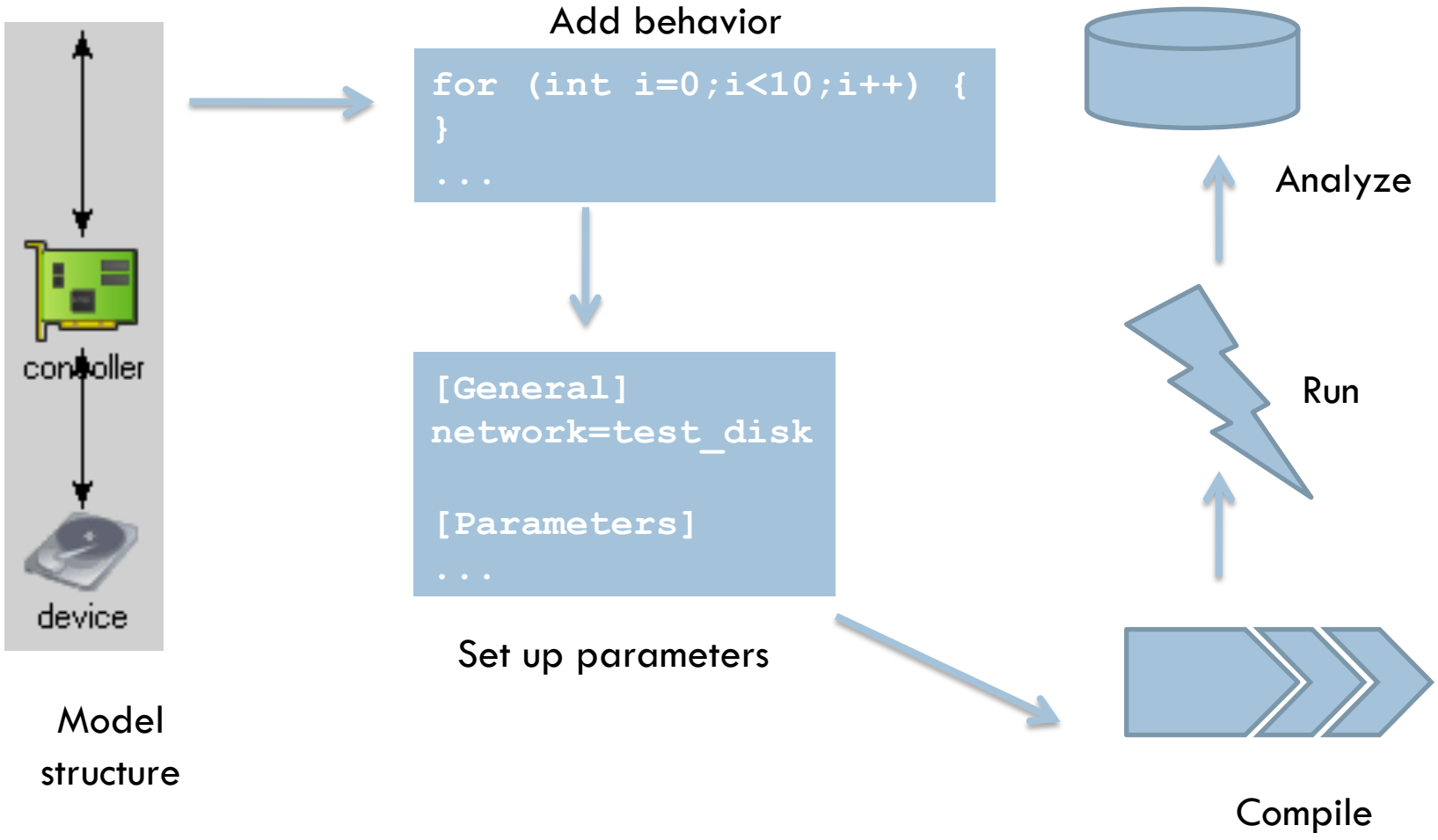
- Documentation system for models is doxygen based and connects very well model documentation and API documentation.



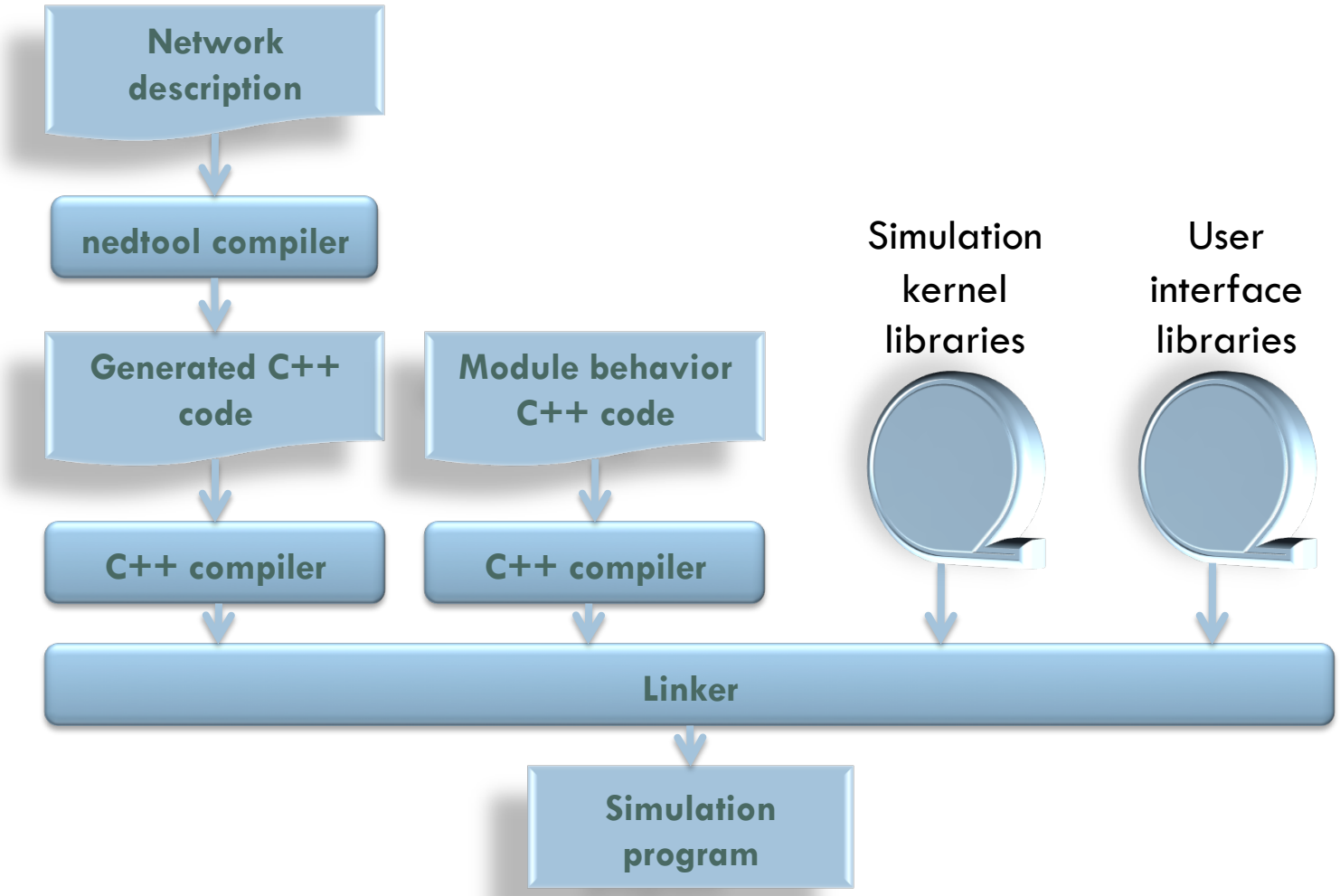
Documentation

The documentation is well organized, automatically generated from source code and easy to browse.

Simulation Model building



Build process

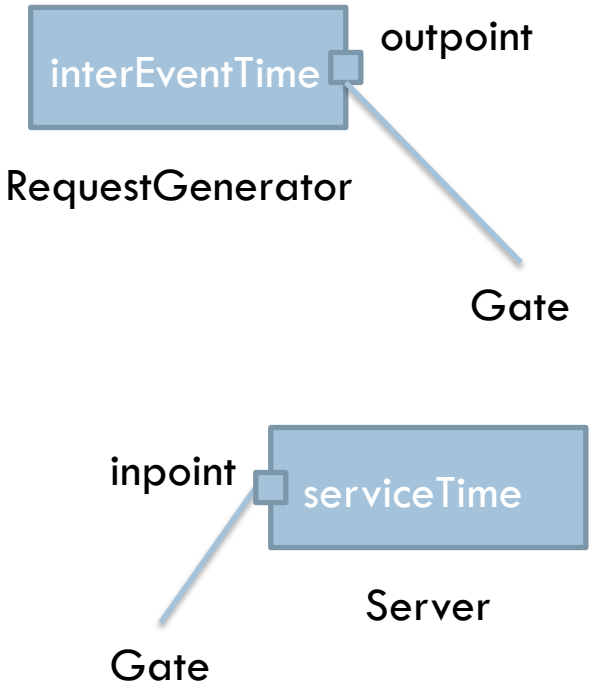


Example

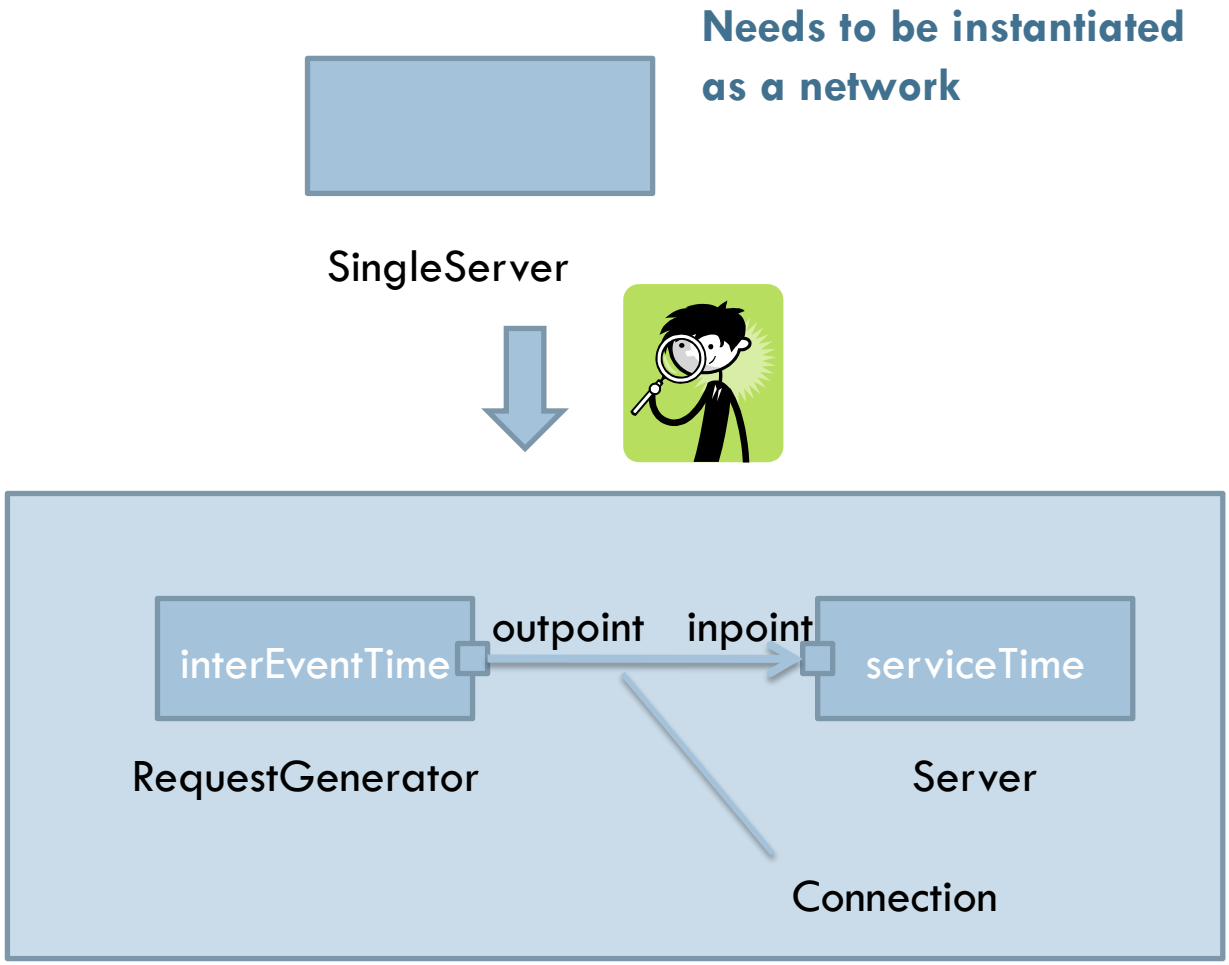
- To show capabilities of OMNET++ basic features.
- Evaluate M/M/1 Queue System.

Step 1: Define the modules

- Two simple modules
  - RequestGenerator.
  - Server.
- Parameters:
  - Set up by an enclosing module or at run-time.



Step 1: Compound module



Step 1: Topology definition

- The whole topology definition is expressed in the specific purpose NED language.
  - Only structural description.
  - Very simple.
  - GUI for NED editing (I prefer text editing).
  - C++ code automatically generated from NED.

Step 2: Add Behavior

- Each simple module needs a C++ class implementing its behavior.
  - Follow user's manual guidelines.
  - Few methods needed.

- No C++ coding for compound modules.

Step 2: Writing the code

- For each simple module → Write a class derived from cSimpleModule.
- Special rules:
  - Two stage creation: Constructor + initialize().
  - Two stage destruction: finish() + Destructor.
  - Message handling:
    - Synchronous (More memory consuming, easy to understand).
    - Asynchronous (Event based, more efficient).

Step 2: Writing the code

RequestGenerator

- Send messages.
- Wait interEventTime between two sending.
- Use a timer to schedule next sending.

Server

- Each time a request arrives process or enqueue.
- Wait serviceTime for processing by using a timer.
- Record statistics of service time.

Step 2: Coding effort

- Ned files: 35 lines
  - RequestGenerator.ned: 6
  - Server.ned: 6
  - SingleServer.ned: 23
- C++ code: 152
  - RequestGenerator.h: 23
  - RequestGenerator.cc: 33
  - Server.h: 31
  - Server.cc: 65

Coding 187 lines

Step 3: Writing a configuration file

- Most things can be set-up via an ini file
  - Changing things without recompiling.
  - Very simple syntax.

- Some things:
  - Model to be run.
  - Time limitation.
  - Random number generator.
  - Parameters.

Step 4: Build

- Utilities for automatically generating makefiles from existing sources.

- You can build a command-line or a GUI-based binary.

Step 5: Running the simulation

- Example 1: constantParams.ini
- Example 2: exponentialParams.ini
- Example 3: batchRuns.ini

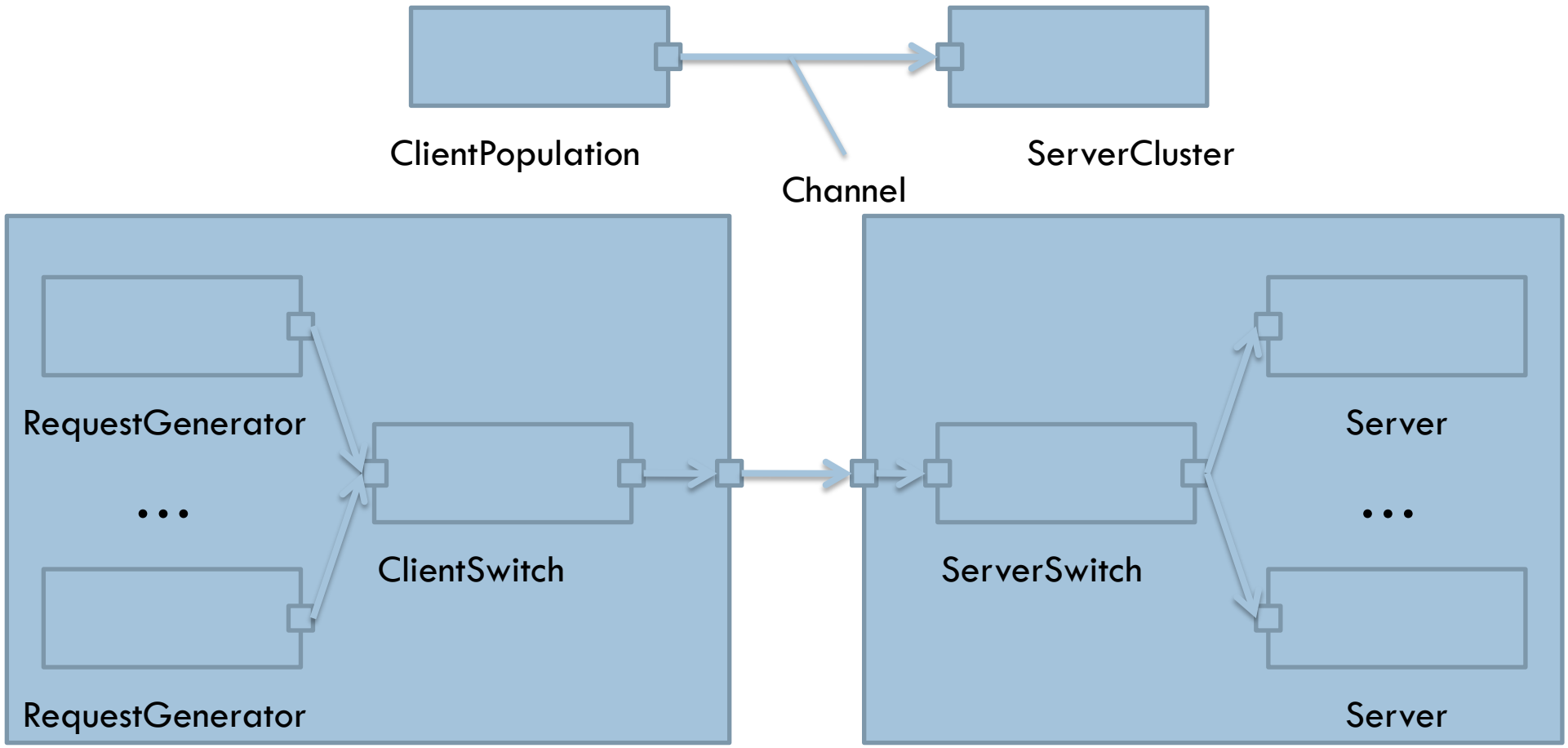
Channels

- It is possible to define a channel and assign it to a link.

- Channel properties:
  - Delay: Propagation delay in seconds.
  - Error: Bit error rate.
  - Datarate: Bandwidth used to calculating transmission time of a message.

- A length must be assigned to the channel.

Example: N Clients to M Servers



Building the simulation

- Step 1: Write the new ned files.
- Step 2: Write/modify code.
- Step 3: Write configuration file
  - Set up number of clients and servers.
- Step 4: Build
- Step 5: Run
  - A simulation with 100000 clients and 40 servers run on a laptop.

Messages

- Messages are sent among modules.

- Attributes:
  - Name: String used in the GUI.
  - Kind: Numeric value representing information type.
  - Length: Number of bits used to compute transmission times.
  - Sending and arrival times.
  - Source and destination moduel and gate.

Message definition language

- In many cases a message needs to carry additional information.
- The structure of the message may be defined in an msg file and then automatically generate C++.

Collecting data and statistics

- Several classes to collect data and compute statistics.
  - Basic statistic estimation.
  - Weigthed statistic.
  - Histograms.
  - Quantiles computation without storing data.
- Transient and accuracy detection at run-time.
- Recording scalars at the end of the simulation.
- Recording vector of data during the simulation.
- Simple view
  - Scalars
  - Plove

Debugging: Watches and snapshots

- During run-time it is possible to watch values from C ++ variables or structures (even modify).
  - Easy to use → Just a macro line.
- Snapshots allow dumping all the objects to a file for debugging the simulation.

INET Framework

- A library of OMNET++ modules for building network simulations.

- Layers:
  - Applications: Protocol specific applications (e.g. Telenet)
  - Mobility: Motion patterns of mobile objects
  - Nodes: Nodes in the network (e.g. Router, StandardHost)
  - Transport: Transport protocols (e.g. TCP)
  - Network: Network protocols (e.g. IPv4)
  - NetworkInterfaces (e.g. Ethernet)

Network interfaces

- Ethernet
- PPP
- IEEE 802.11

Network

- Protocols
  - ARP.
  - ICMP (v4 & v6).
  - IP (v4 & v6).
  - LDP.
  - MLPS.
  - OSPF Routing.
  - RSVP.

- Other functionality
  - Automatic network configuration.

Transport

- TCP
- UDP
- RTP

Nodes

- Contains nodes for different types of protocols.
  - Mainly routers and hosts.

- IPv4.
- IPv6.
- MPLS.
- Wireless.
- Ad-hoc networks.

Applications

- Generic applications for
  - Ethernet
  - TCP
  - UDP
  - Ping.
- Client and server applications.

Examples

- Example 1: Web server
  - Define the network topology.
  - Configure parameters.
  - Run.
- Simple changes
  - Number of clients.
  - NAM tracing.

OMNeT++ Discrete Event Simulation System

INTRODUCTION TO SIMULATION WITH **OMNET++**

José Daniel García Sánchez
ARCOS Group – University Carlos III of Madrid