

# 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++?

3

- 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

4

- 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

5

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

# Randomness

6

- 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

7

- 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

8

- 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.

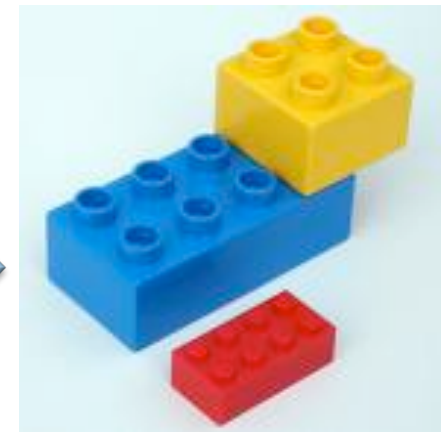


# Model management

9

- 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.

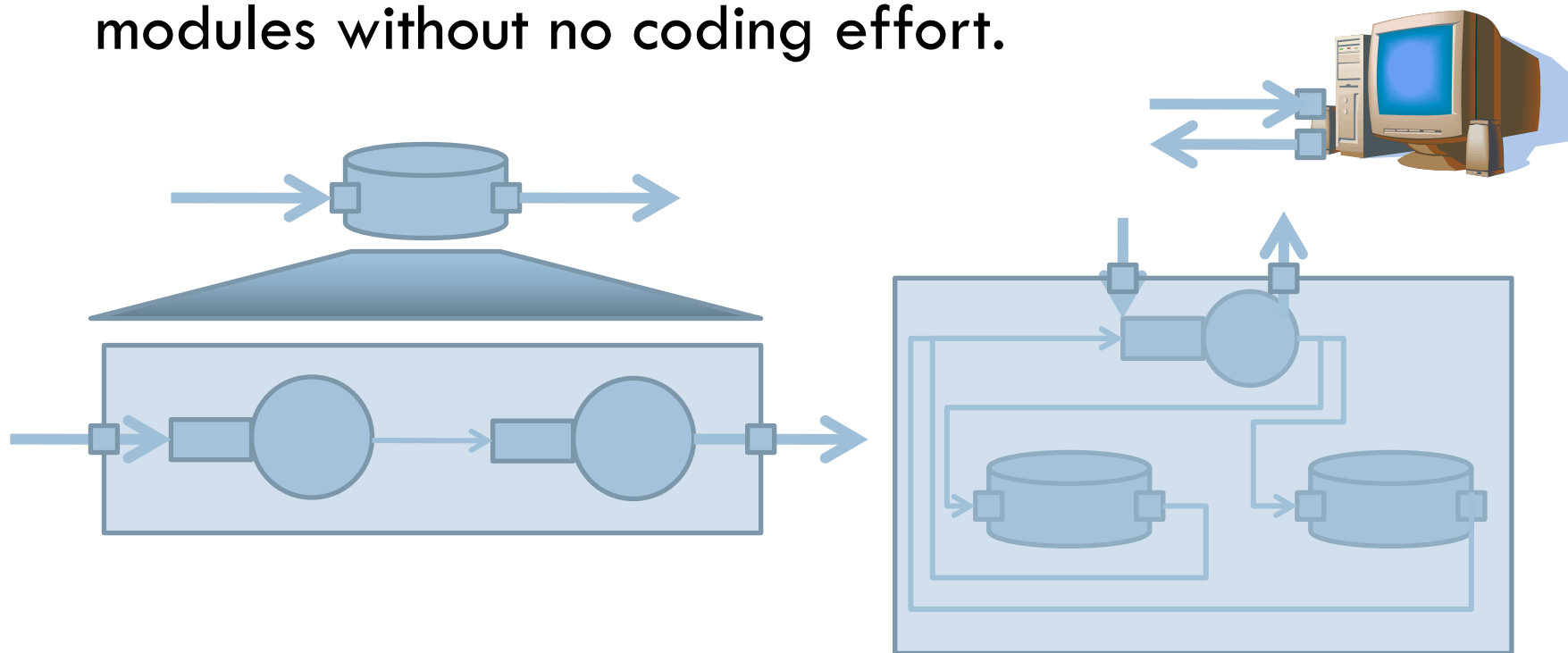
Build models and combine like  
**LEGO blocks**



# Hierarchical modeling

10

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



# Debugging and tracking

11

- 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

12

- 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.

Model documentation -- generated from NED files - Mozilla Firefox

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

http://www.omnetpp.org/doc/INET/neddoc/index.html

Customize Links Free Hotmail Windows Marketplace Windows Media Windows WITS Google Calendar FI-UPM :: Indice Storage Performance... Noticias de la CODDI ... STAR Group Main Page - CS411

Model documentation -- generat...

- [SnrEval60211](#)
- [SnrNic](#)
- [StandardHost](#)
- [StandardHost6](#)
- [TCP](#)
- [TCPApp](#)
- [TCPBasicClientApp](#)
- [TCPDump](#)
- [TCPEchoApp](#)
- [TCPGenericSrvApp](#)
- [TCPSessionApp](#)
- [TCPSinkApp](#)
- [TCPSpooF](#)
- [TCPSpooFingHost](#)
- [TCPSrvHostApp](#)
- [TED](#)
- [TelnetApp](#)
- [ThruputMeter](#)
- [TurtleMobility](#)
- [UDP](#)
- [UDPApp](#)
- [UDPBasicApp](#)
- [UDPEchoApp](#)

**NED and MSG Files**

- [Applications/Ethernet/EtherApp.msg](#)
- [Applications/Ethernet/EtherAppCli.ned](#)
- [Applications/Ethernet/EtherAppSrv.ned](#)
- [Applications/Generic/PTrafGen.ned](#)
- [Applications/Generic/PTrafGenerator.ned](#)
- [Applications/PingApp/PingApp.ned](#)
- [Applications/PingApp/PingPayload.msg](#)
- [Applications/TCPApp/GenericAppMsg.msg](#)
- [Applications/TCPApp/TCPApp.ned](#)
- [Applications/TCPApp/TCPBasicClientApp.ned](#)
- [Applications/TCPApp/TCPEchoApp.ned](#)

## Simple Module TCP

**File:** [Transport/TCP/TCP.ned](#)

**C++ definition** [click here](#)

TCP protocol implementation. Supports RFC 793, RFC 1122, RFC 2001. Compatible with both IPv4 and [IPv6](#).

A TCP segment is represented by the class [TCPSegment](#).

**Communication with clients**

For communication between client applications and [TCP](#), the [TcpCommandCode](#) and [TcpStatusInd](#) enums are used as message kinds, and [TCPCommand](#) and its subclasses are used as control info.

To open a connection from a client app, send a cMessage to [TCP](#) with TCP\_C\_OPEN\_ACTIVE as message kind and a [TCPOpenCommand](#) object filled in and attached to it as control info. (The peer [TCP](#) will have to be LISTENING; the server app can achieve this with a similar cMessage but TCP\_C\_OPEN\_PASSIVE message kind.) With passive open, there's a possibility to cause the connection "fork" on an incoming connection, leaving the original connection LISTENING on the port (see the fork field in [TCPOpenCommand](#)).

The client app can send data by assigning the TCP\_C\_SEND message kind and attaching a [TCPSendCommand](#) control info object to the data packet, and sending it to [TCP](#). The server app will receive data as messages with the TCP\_I\_DATA message kind and [TCPSendCommand](#) control info. (Whether you'll receive the same or identical messages, or even whether you'll receive data in the same sized chunks as sent depends on the sendQueueClass and receiveQueueClass used, see below. With TCPVirtualDataSendQueue and TCPVirtualDataRcvQueue set, message objects and even message boundaries are not preserved.)

To close, the client sends a cMessage to [TCP](#) with the TCP\_C\_CLOSE message kind and [TCPCommand](#) control info.

[TCP](#) sends notifications to the application whenever there's a significant change in the state of the connection: established, remote [TCP](#) closed, closed, timed out, connection refused, connection reset, etc. These notifications are also cMessages with message kind TCP\_I\_XXX (TCP\_I\_ESTABLISHED, etc.) and [TCPCommand](#) as control info.

One [TCP](#) module can serve several application modules, and several connections per application. The kth application connects to [TCP](#)'s from\_app[k] and to\_app[k] ports. When talking to applications, a connection is identified by the (application port index, conId) pair, where conId is assigned by the application in the OPEN call.

**Sockets**

Terminado

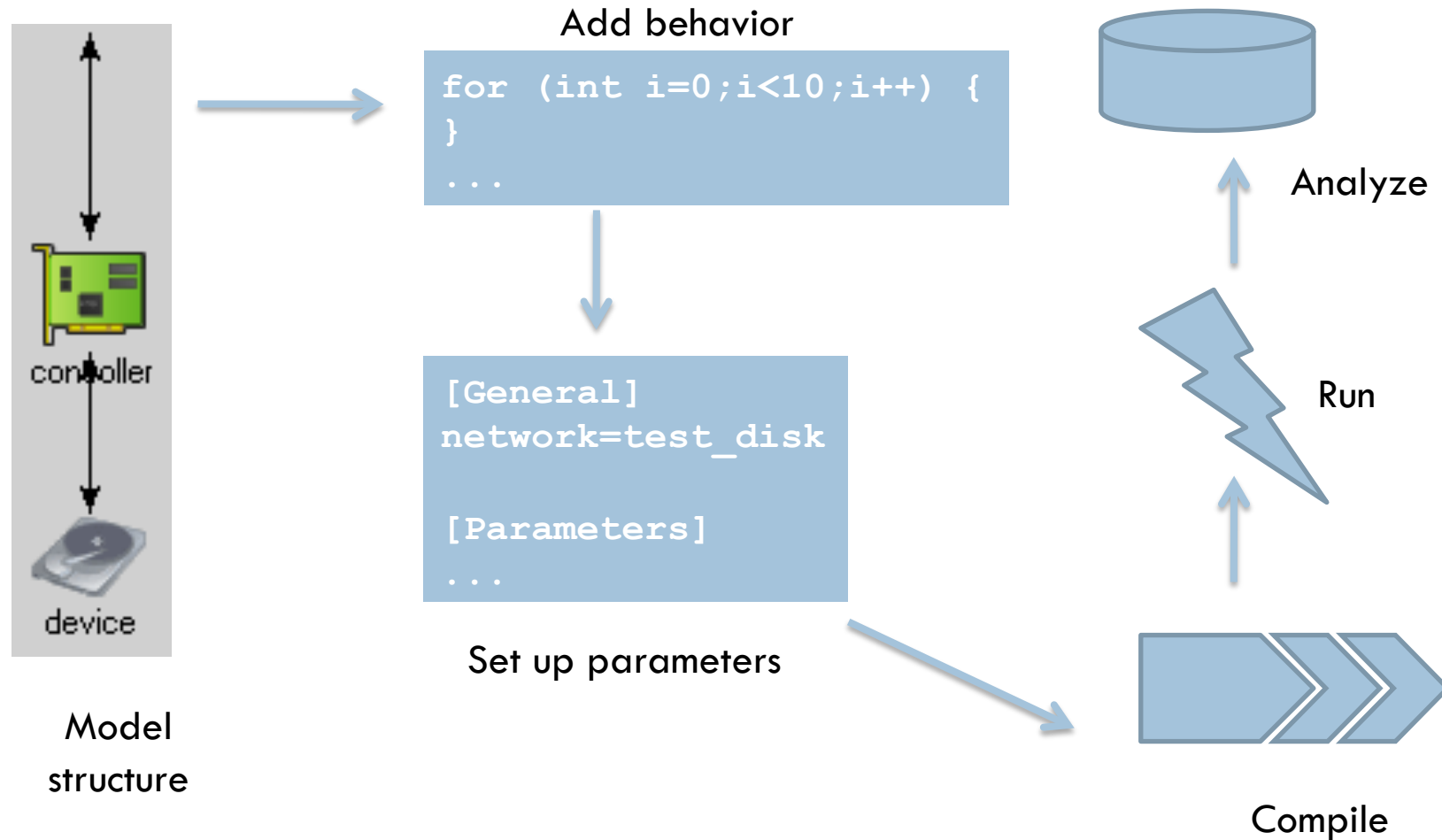
Inicio

15:13

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

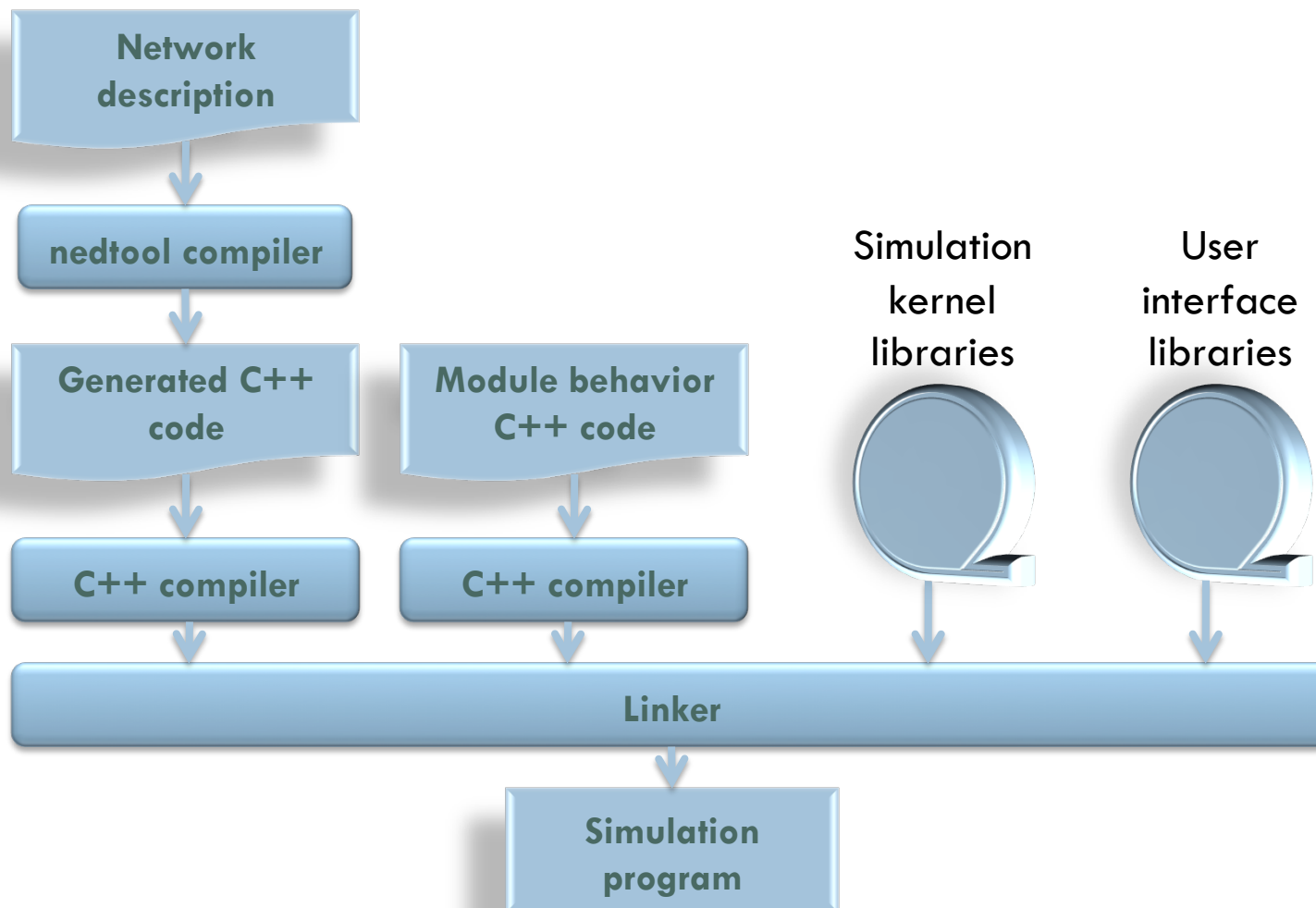
# Simulation Model building

14



# Build process

15



Introduction to simulation with OMNET++

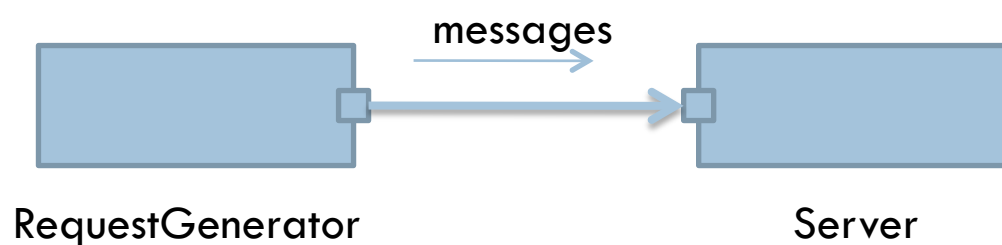
José Daniel García Sánchez – ARCOS Group – University Carlos III of Madrid

July 2007 - University of Modena

# Example

16

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

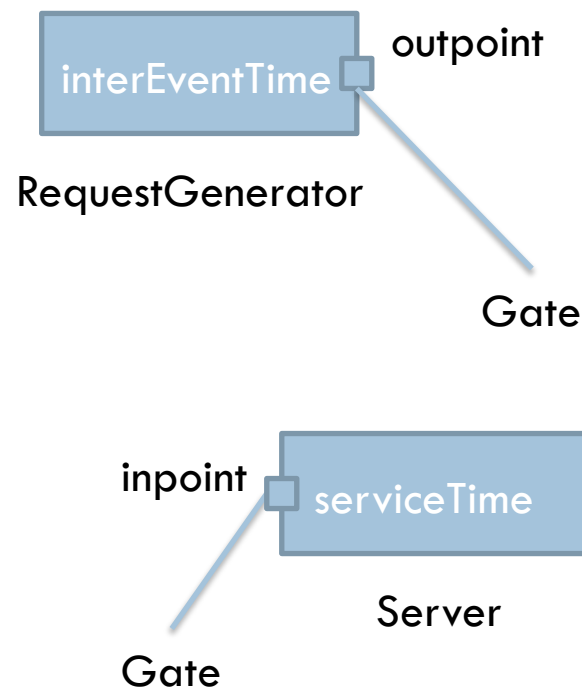




# Step 1: Define the modules

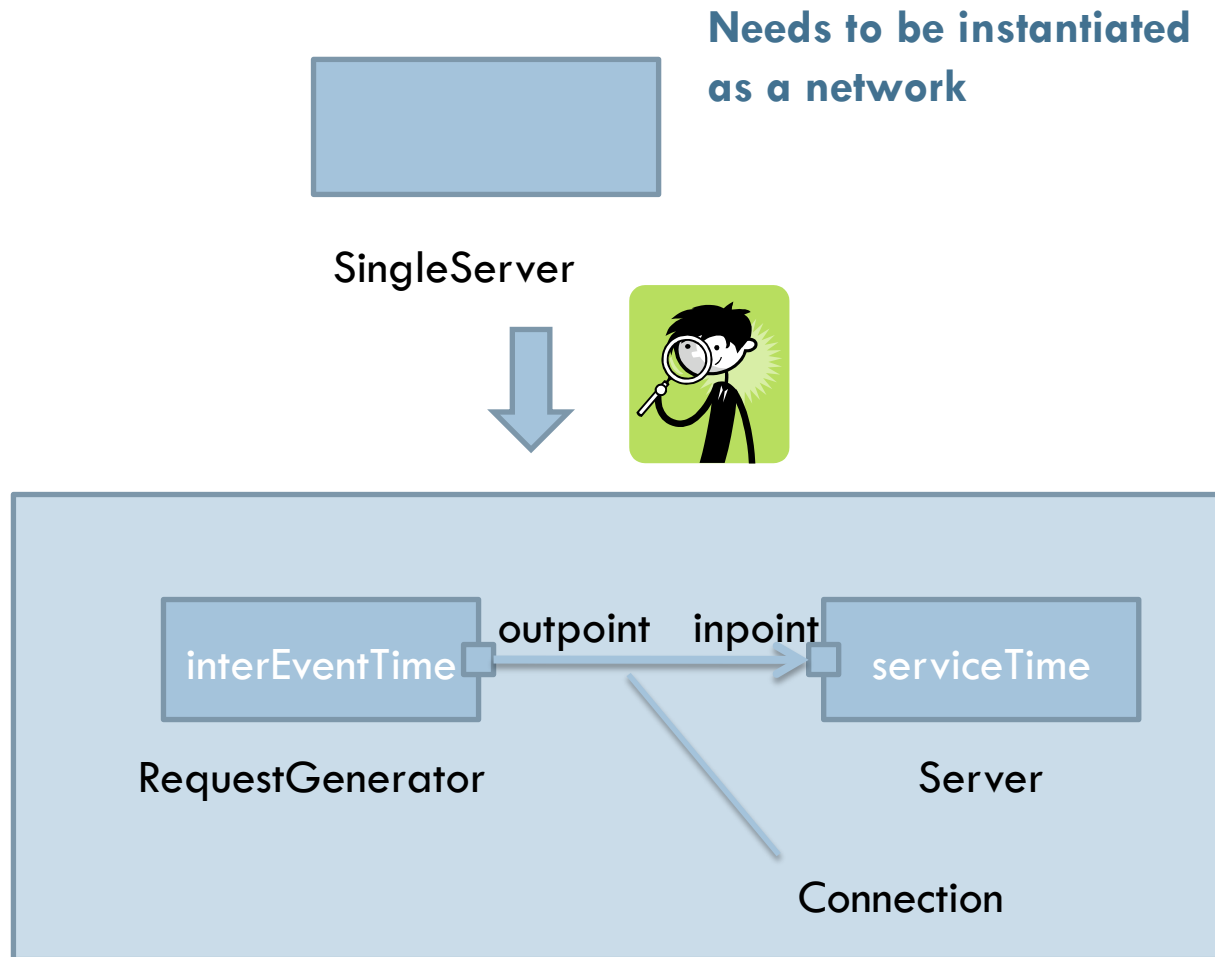
17

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



# Step 1: Compound module

18



# Step 1: Topology definition

19

- 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

20

- 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

21

- 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

22

## 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

23

- 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

24

- 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

25

- Utilities for automatically generating makefiles from existing sources.
- You can build a command-line or a GUI-based binary.

# Step 5: Running the simulation

26

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

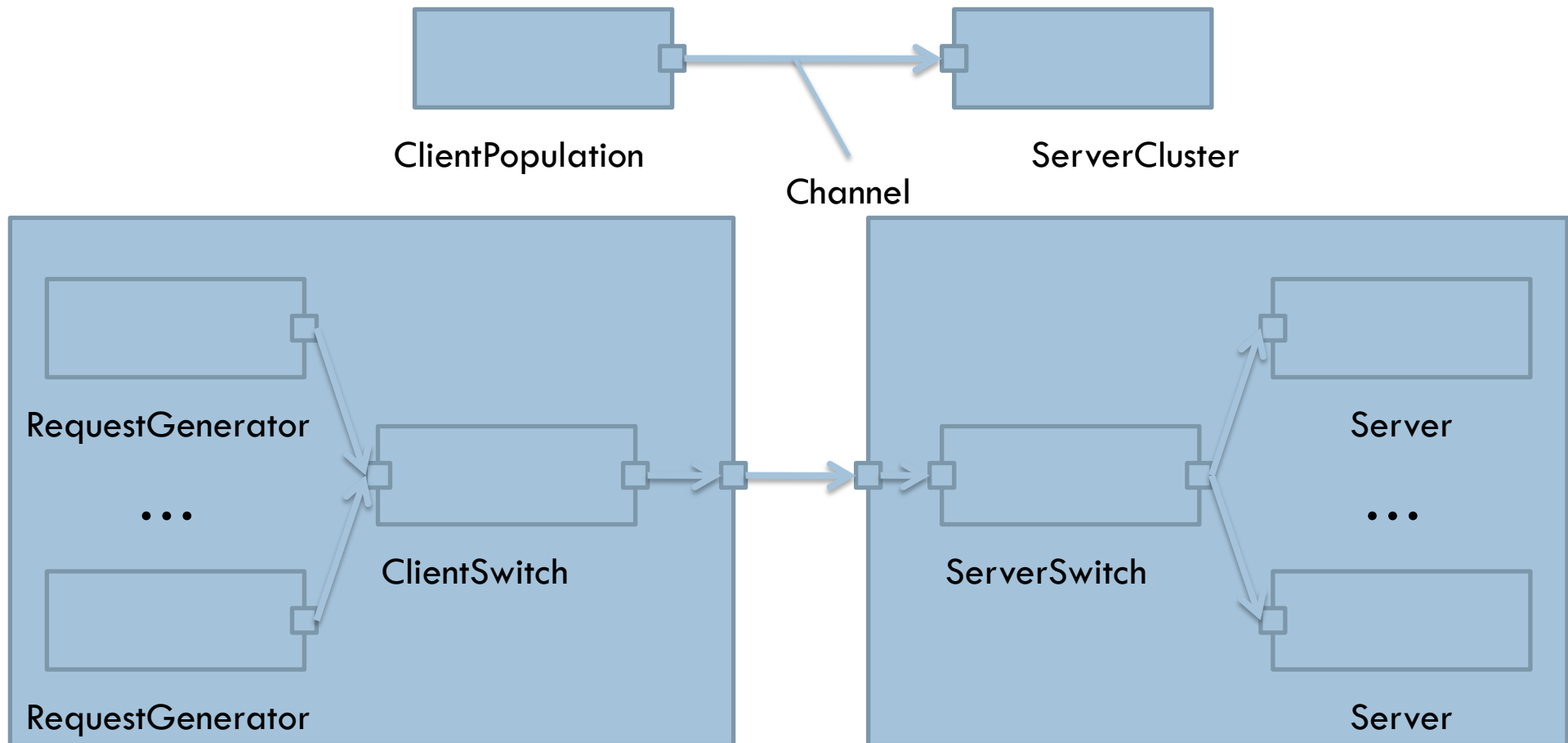
# Channels

27

- 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

28



# Building the simulation

29

- 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 1 000 000 clients and 40 servers run on a laptop.

# Messages

30

- 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 module and gate.

# Message definition language

31

- 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

32

- Several classes to collect data and compute statistics.
  - ▣ Basic statistic estimation.
  - ▣ Weighed 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

33

- 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

34

- 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

35

- Ethernet
- PPP
- IEEE 802.11

# Network

36

- Protocols
  - ARP.
  - ICMP (v4 & v6).
  - IP (v4 & v6).
  - LDP.
  - MLPS.
  - OSPF Routing.
  - RSVP.
  
- Other functionality
  - Automatic network configuration.

# Transport

37

- TCP
- UDP
- RTP

# Nodes

38

- Contains nodes for different types of protocols.
  - Mainly routers and hosts.
  
- IPv4.
- IPv6.
- MPLS.
- Wireless.
- Ad-hoc networks.

# Applications

39

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

# Examples

40

- 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

