Global Journal of Computer Science & Technology
Volume 11 Issue 6 Version 1.0 April 2011
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

The Analysis And Implementation Of OMNeT++ Framework

By K. Sivakumar , G. Dalin

Abstract- : The OMNeT++ is basically a collection of software tools and libraries which you can use to build your own simulation models. OMNeT++ is an object oriented modular discrete event network simulation framework. It has a generic rchitecture, so it can be used in various problem domains, such as, validating of hardware architecture, evauating performence aspects of the complex software systems, protocol modelling etc.. OMNeT++ simulations can be run under various use interfaces. When building and running the OMNeT++ simulation we must consider the topology, messages and the simulation system provides the simulation kernel and user interfaces. This paper presents the implementation and analysis of OMNeT++ framework, visualizing results with plove and scalars.

Keywords: *Simulation, NED language, modules, parameters, messages, analysis, IDE, simulation kernel, discrete, eventlog, own IDE modules & packet.*

Classification: *GJCST Classification: C.2.4*

Strictly as per the compliance and regulations of:

The Analysis and Implementation of OMNeT++ Framework

K. Sivakumar[α], G. Dalin[Ω]

Abstract- The OMNeT++ is basically a collection of software tools and libraries which you can use to build your own simulation models. OMNeT++ is an object oriented modular discrete event network simulation framework. It has a generic rchitecture, so it can be used in various problem domains, such as, validating of hardware architecture, evauating performence aspects of the complex software systems, protocol modelling etc.. OMNeT++ simulations can be run under various use interfaces. When building and running the OMNeT++ simulation we must consider the topology, messages and the simulation system provides the simulation kernel and user interfaces. This paper presents the implementation and analysis of OMNeT++ framework, visualizing results with plove and scalars.

Keywords: Simulation, NED language, modules, parameters, messages, analysis, IDE, simulation kernel, discrete, eventlog, own IDE modules & packet.

I. Introduction

The OMNeT++ IDE is based on the Eclipse platform which is an extensible, Java based framework. While it started as an IDE framework only, its main goal is to be a generic integration platform. OMNeT++ adds functionality for creating and configuring models (NED and ini files), performing batch executions, and analyzing simulation results, while Eclipse provides C++ editing, SVN/GIT integration, and other optional features via various open-source and commercial plug-ins. The OMNeT++ IDE is in fact an Eclipse istallation with some additional - simulaton related - tools pre-installed:

• The OMNeT++ feature which contains all OMNeT++ specific tools you use: the NED, MSG and INI file editor, simulation launcher, result analysis tools, sequence char view, documentation generator etc.

• CDT (C/C++ Development Tooling - eclipse.org/cdt) - for C++ development and debugging. This feature integrates with the standard gcc toolchain and the gdb debugger.

If you would like to develop your own plugins for the IDE you will need to install some additional components manuall

* JDT (Java Development Tools) for java development. JDT contains a java compiler and all the editors and debuggers and tools used during java development.

* PDE (Plug-in Development Environment) - this component contains additional tools. API definitions and documentation for developing plugins. PDE requires the presence of JDT.

a) The NED Editor

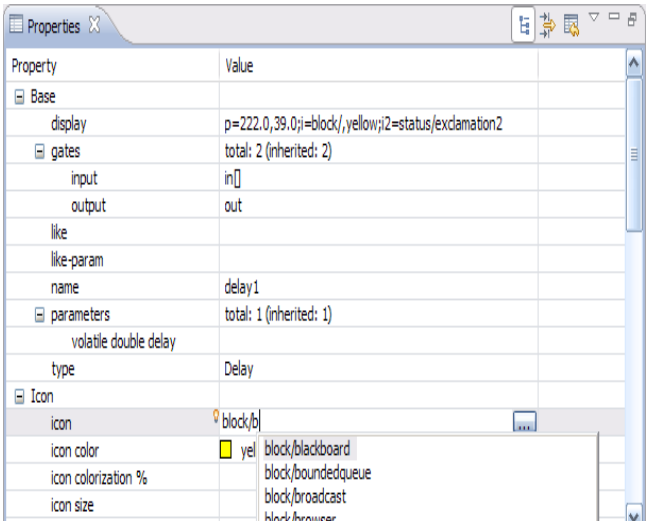
The NED Editor can edit NED files both graphically or in text mode, and the user can switch between two modes at any time, using the tabs at the bottom of the editor window In graphical mode, one can create compound modules, channels and other component types. Palettes are used to submodules from available module types.

Properties view is used to modify the visual and non-visual properties of the context menu. The NED Editor provide the following graphical features,

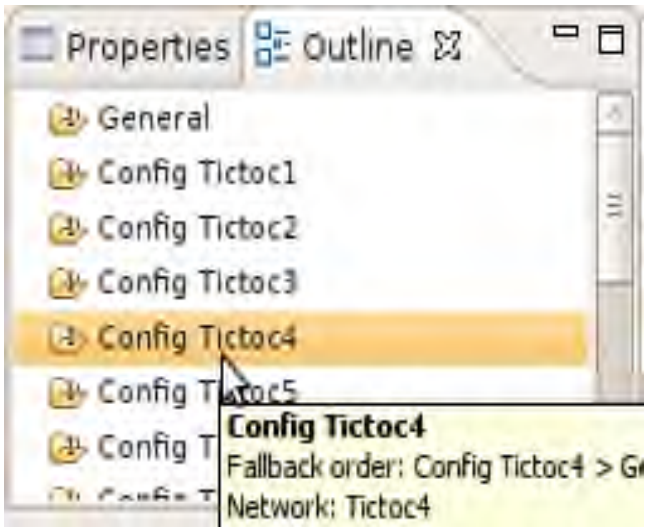
* Editing background image.
* Editing background grid.
* Default icons.
* Icon sizing and coloring.
* Transmission range and many others.

The properties view lets the user edit graphical and non- graphical properties of objects. Special cell editors facilitate selecting colors, icons etc., Undo and Redo is supported for property changes too. The properties view is also used with other editors like the Result analysis editor, where it allows the user to customize charts and other objects. The NED source is continually parsed and validated as the user is, typing and errorsare displayed in real time on the left margin. Syntax highlighting automatic indentation, and automatic conversion from the OMNeT++ NED syntax are also provided. NED parameters view are also used in NED editor.

About[α] - M.Sc., MCA., M.Phil, Assistant Professor SNMV CAS
E-mail- ksivadreams@gmail.com.
About[Ω] - M.SC(SS)., Assistant Professor SNMV CAS
E-mail- gdalinmsc@gmail.com



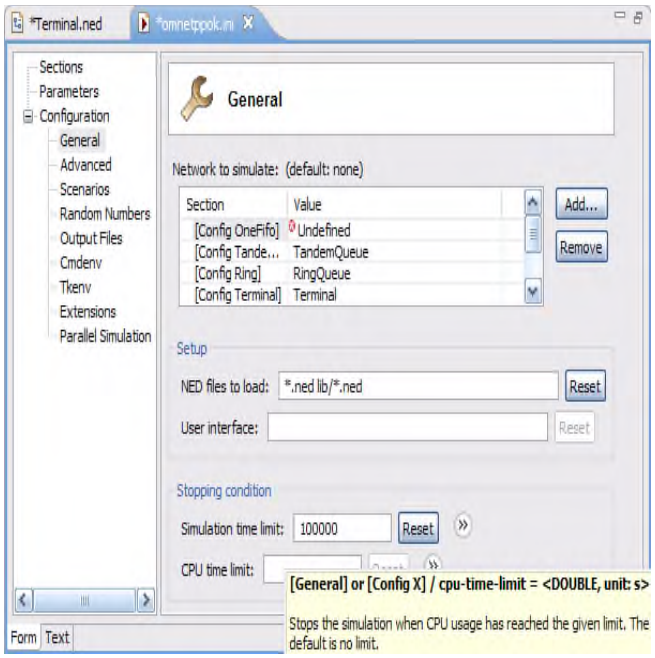
Properties View



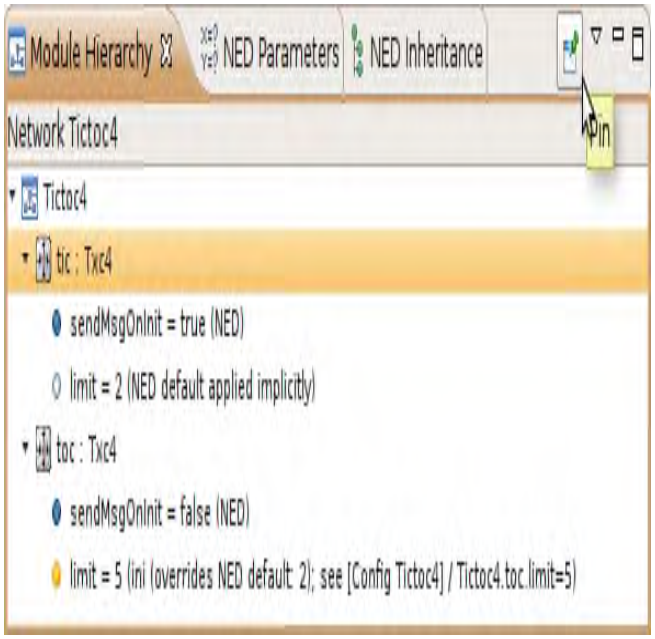
Outline View

b) *The INI File Editor*

The Ini File editor is used to support for user to configure simulation models for execution. The Ini Editor is used for both form based and source editing. The structure of the ni file is visualized and editable using Drag and drop and dialogs. The user can easily work under the text editor. The content of the INI file is divided into sections.



Form based INI File Editing



Module Hierarchy View

On the first page of the form editor, we can edit the sections. The sections are displayed as a tree, the nodes inherit the settings from parent. The INI file editor considers all supported configuration options and offers them in several forms, organized by topics.

II. Building OMNeT++ Application

Various views are available in INI Editor. These views are displayed by choosing the view from window.

1. *Outline View*

It provide the overview of sections in the current INI file.

2. *Problem View*

The problem view is used to view the errors and warning messages. The parser is generate these kinds of messages.

3. *Module Hierarchy view*

It shows the submodules, module parameters and where its values comes from.

4. Parameters view

The selected section parameters are displayed in this view. It also displays the inherited parameters and unassigned in the configuration.

5. NED Inheritance view

It display the inheritance tree of the network configured in the selected section.

These views can be displayed by choosing the view from window, then select view submenu.

a) The NED Language

OMNeT++ is an object oriented modular discrete event simulator. The abbreviation of OMNeT++ is "Objective Modular Network Testbed in C++". Each entity in a simulation needs to communicate via messages with itself and other entities. messages can be used for many purposes. One is to represent sticks that are available. Another one is to convey information of the entity. Simple modules are defined in NED file by their i) Parameters ii) Gates.

* Parameters:

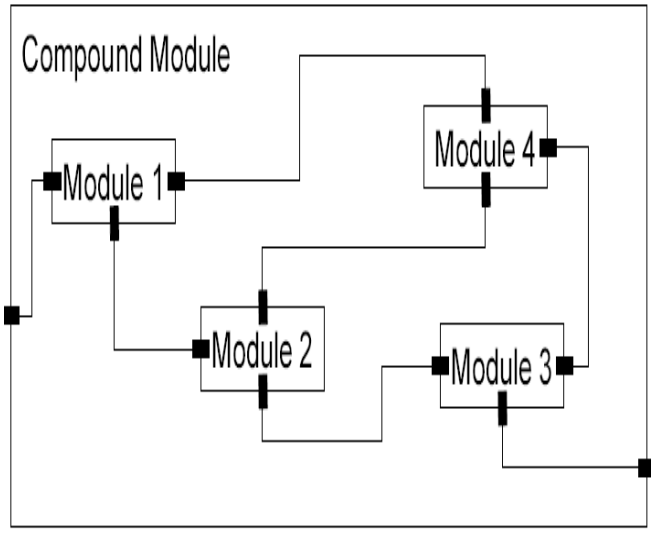
Values that can be set from outside the simulation program. Parameters can be easily accessed from the C++ code using cModule's method

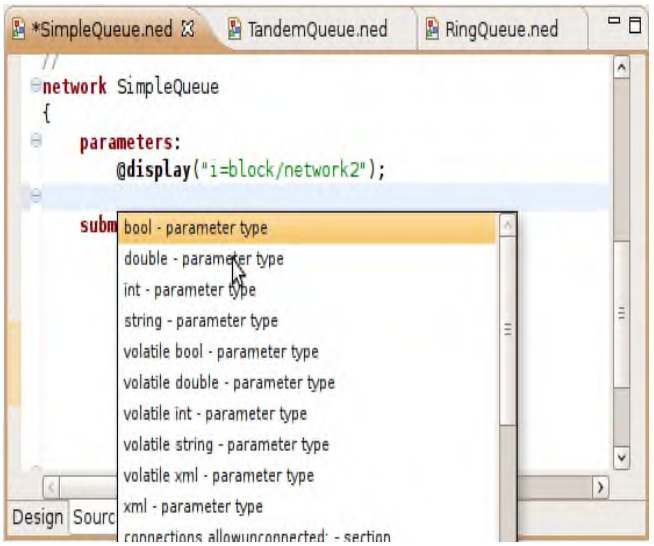
* Gates:

In NED, the gates of simple modules are defined as well as either in or out gates.

Gates can also be defined as arrays. Gates encapsulate the knowledge where to within the module.

The NED compound modules consist of one or more sub modules. It has two things such as Inside & Outside. Gates of the compound modules are connected to gates of the composing modules. Parameters of compound modules are similar to simple modules. Gates of the compound modules are identical to simple module gates.





NED Functions in Editor

Submodules section of a compound module defines which modules constitute the compound module. Submodules can be written as vectors of modules. Module type of sub modules need not be specified explicitly, can be left as a parameter, but the interface must be declared. The NED language describes links among modules and it also describes composition model. It has two divisions, it may be text edited or GUI.

b) NED Functions

The following functions are used in NED expressions and INI files. The NED functions are classified into various categories. The categories are,

c) Conversion

Conversion functions are used to convert on type to other. Such as double, int and string.

d) Math

For mathematical calculations, we using the following functions.

1. fabs: Returns the absolute value of the quantity.
2. fmod: Returns floating-point remainder.
3. max: Returns the greater one of two quantities.
4. min: Returns the smaller one of two quantities. Trigonometric functions also used in math NED.

e) NED

More number of functions used in NED, and it is based on the module or channel.

1. ancestorIndex: Returns the index of the ancestor module.
2. FullName: Returns the full name of the module.
3. Fullpath: Returns the full path the module.
4. ParentIndex:Returns the index of the parent module.

The other NED function categories are as follows,

f) random/continuous

g) random/disrete

The NED language features are as folloows,

1. Hierachical :- A complex single entity is broken into smaller modules, and used as a compound module.
2. Compoenet Based Module:- These modules are inherently reusable and it allows component libraries.
3. Interfaces:- Interfaces module can be used as a placeholder where normally a module or channel type would be used.
4. Packages:- The NED language features a java-like package structure, to reduce the risk of name clashes between different models.

h) Modules

A complex problem or large peoblem is divided into smaller units is called Modules. Basically modules are used in two major reasons, such as, 1. To reduce the length of the code. 2. To increase the execution speed.

A discrete event system is a system where state changes happen at discrete instances in time, and events take zero time to happen. It is assumed that nothing happens between two consecutive events, that is, no state take place in the system between events. The time when event occur is often called event timestamp. With OMNeT++ we use the term arrival time. Time within the model is often called simulation time, model time or virtual time.

1. Events in OMNeT++

OMNeT++ uses messages to represent events. Each event is represented by an instance of the cMessage class or one of its subclasses; there is no separate event class. Messages sent from one module to another-this means that the place where the "event will occur" is the message's destination module, and the model time when the event occurs is the arrival time of the message.

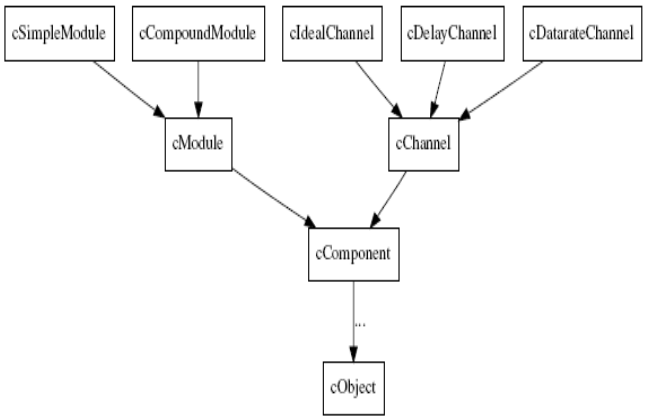
Events like "timeout expired" are implemented by the module sending a message to itself. Events are consumed from the FES in arrival time order, to maintain causality. More precisely, given two messages, the following rules apply:

* The one with the smaller scheduling priority value is executed first. If priorities are the same,

* The one scheduled or sent earlier is executed first. Add Scheduling priority is a user-assigned integer attribute of messages.

2. Components, Simple Modules and Channels

OMNeT++ simulation models are composed of modules and connections; Modules may be simple modules or compound modules. Simple modules are the active components. connections may have assoociated channel objects. Channel object encapsulate channel behavior, propagation and transmission time modeling, error modeling. Modules and Channels are called components.Components are represented with the c++ class cComponent. Simple Modules has two subclasses called cSimpleModule and cCompoundModule.



III. Implementation

The discrete event simulation can be divided into three main components, such as

1. *Core Platform.*
2. *Protocol Library.*
3. *Documentation.*

The main platform include lot of functionalities, based on the service. The major functionalities are,

1. *User Interface.*
2. *Simulation Engine.*
3. *Result Alalysis Tool.*

The network simulation is done using the following steps

1. *Set up the topology.*
2. *Configure protocol parameters.*
3. *Set traffic pattern.*
4. *Run simulation.*
5. *Analyze the results.*

The core platforms two major functionalities are, User Interface may be graphical user interface (GUI), textual and command line or hybrid. Simulation engine is the unit that executes the discrete event simulation. It stores the event list and manipulates it.

a) Creating Projects

The OMNeT++ IDE contains CDT to help you develop the C++ part of your simulation. The OMNeT++ IDE several extensions to the standard CDT features for ease of use:

* Customized project creation dialog.

* Extended C++ toolchain for OMNeT++

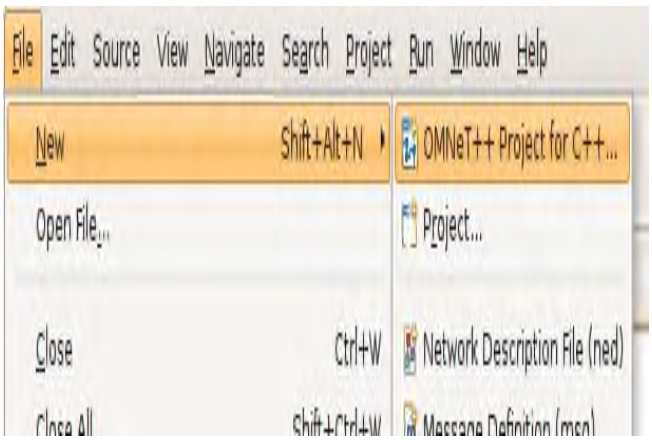
* Automatic makefile generation.

* OMNeT++ specific build configuration.

* Special launch configuration for launch & debug

1. Creating a C++ project

To create an OMNeT++ project that supports C++ development. Select New option from file menu. The following dialog box show the new OMNeT++ project wizard.



This will show the new OMNeT++ project wizaed that lets create a project that supports NED, MSG and INI file editing as well as C++ development for simple modules.

2. Configuring the Project

The OMNeT++ IDE adds several extensions to the standard CDT to make the building of simulations much easier. The project properties dialog is extended with a special page - OMNeT++ | Makemake - where you can configure the options used for automatically generating the makefile(s). A separate makefile will be created according to the settings you specify in the dialog. On the C/C++ Build page you can specify which makefile will be called by the IDE build process when you initiate a Build action. To configure the makefile the following options will be used.

* The target type can be either executable or a shared/static library.

* You may set the target name.

* The output directory can specify where the object files & the final target will be created.

3. Dependent Project

If our project uses code from other projects we should make the project dependent on that code. This step ensures that our code is linked with the dependent project's output and that the NED path can be correctly set. Include files are also automatically used from dependent projects.

b) Editing C++ code

The OMNeT++ IDE comes with a C++ editor provided by the CDT component. In addition to the standard editor features provided by the Eclipse environment, the C++ editor provides syntax highlighting and content assistance on the source code.

Use CTRL+SPACE to activate the content assist window anywhere in our source code. An other useful key is CTRL+TAB, which switches between your C++ and header file. Press CTRL+SHIFT+L to get a list of currently active key bindings.

c) Building the Project

When we create the source file, it is configured and it is converted into executable format. Once the makefile is configured correctly, it is transferred into project using build project option from the project menu. After the makefile is converted into project, we should switch to the console view to see the actual progress of the build. The project is debugged and we see the progress using the following views,

* Outline view
* Type Hierarchy view
* Problems view
* Console View

IV. Analyzing the Results

OMNeT++ is the statistical analysis tool is integrated into Eclipse Environment. When we creating an analysis, the user first selects the input of the analysis by specifying file names or file name patterns.

Data of interest can be selected into datasets using extra pattern rules. The user can define datasets by adding various processing, filtering and charting steps.

V. Conclusions

We presented a simulator focused application development and simulation point of view. OMNeT++ is the discrete event simulator mainly focused on the communication networks. OMNeT++ is designed to support parallel execution(MPI based), and it runs on both windows and Linux. The key elements of OMNeT++ programminh model is topology and behavior. Topology of module connections is specified using an OMNeT++ specific language called NED. OMNeT++ is a tool for discrete event simulation, managment of events is a primary task. Events are generated by modules sending messages to other modules orthemselves. It supports both finite state/event based simulation as well as process based simulation. OMNeT++ is the best tool for simulating the communication network and various research groups building the different models.

References Références Referencias

1. Gene Bellinger. (2004). Modeling & Simulation – An Introduction URL: http://www.systems-thinking.org/ simulation/model.htm
2. OMNeT++ Home Page. http://www.omnetpp.org
3. MiXiM home page.http://sourceforge.net/projects/mixim
4. H. Karl. Praxis der Simulation. course slides, Technische Universität Berlin.
5. Varga. OMNeT++: Object-Oriented Discrete. Event Simulator http://www.omnetpp.org/
6. David A. Cook. (2001). Computers and M&S - . Modeling and Simulation. The Journal of Defense Software Engineering
7. Patrick J. Delaney. "What is a Simulation? https://www.amso.army.mil/library/primers/what-is/
8. I.Dietrich, C. Sommer, F. Dressler. Simulating DYMO in OMNeT++.
9. S. Valentin. 2006. ChSim - A wireless channel simulator for OMNeT++,
10. Max Caceres, “Syscall Proxying - Simulating remote execution”,
11. Mattias Björlin. (2005) A study of Modeling and Simulation for computer and network security using OPNET
12. JiST home page.http://jist.ece.cornell.edu