OMNEST

Installation Guide

Version 4.5

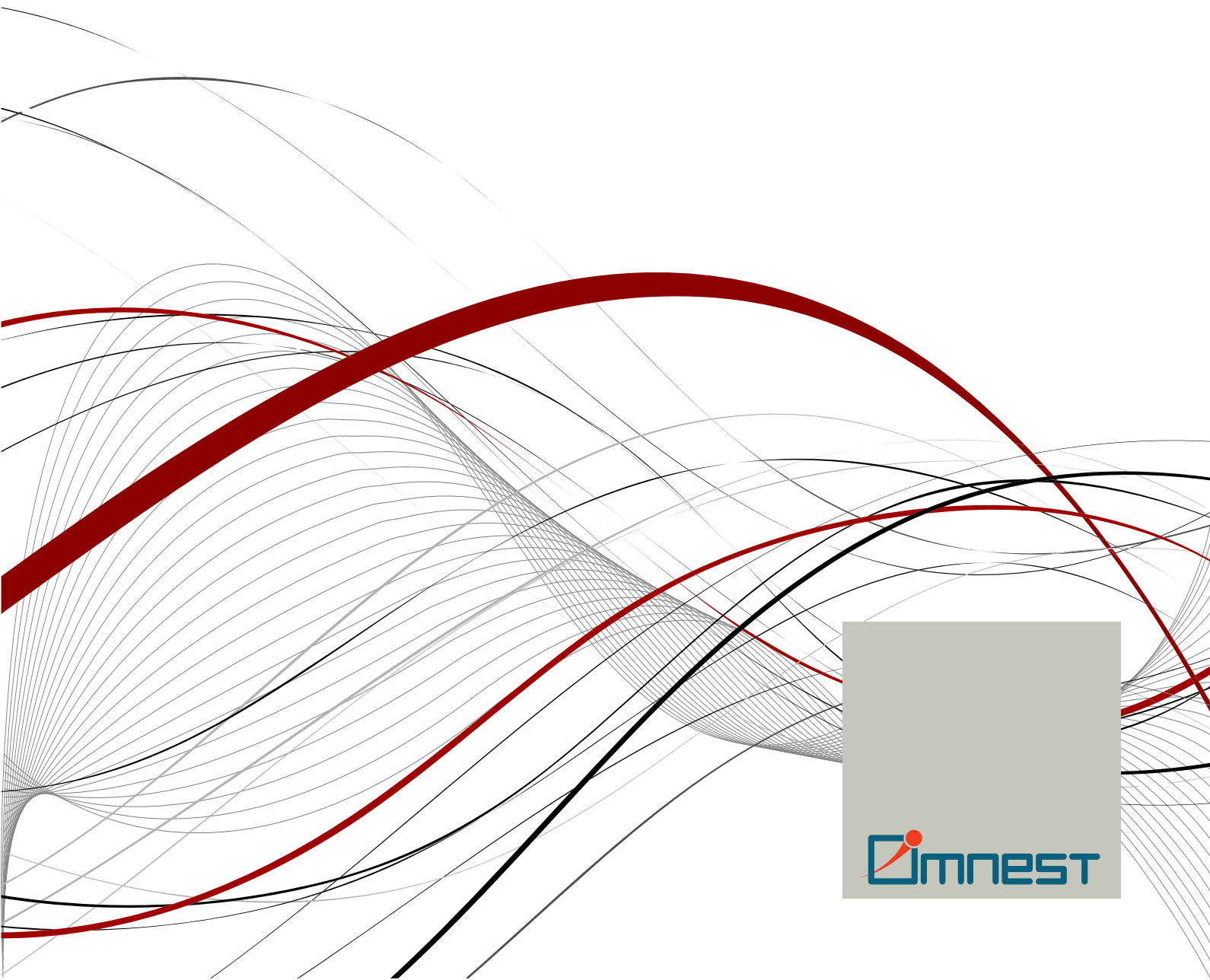

Table of Contents

Chapter 1. General Information

1.1. Introduction

This document describes how to install OMNEST on various platforms. One chapter is dedicated to each operating system.

1.2. Supported Platforms

OMNEST has been tested and is supported on the following operating systems:

- Windows 7, 8 and XP
- Mac OS X 10.7,10.8 and 10.9
- Linux distributions covered in this Installation Guide

32-bit precompiled binaries are provided for the following platforms:

- Windows 7, 8 with Microsoft Visual C++ versions 11 (2012) and 12 (2013)
- Windows 7, 8 and XP with the bundled MinGW GCC 4.x compiler

On other platforms, OMNEST needs to be compiled from source.

The Simulation IDE can be used on the following platforms:

- Linux x86 32/64-bit
- Windows 7, 8 and XP
- Mac OS X 10.7,10.8 and 10.9

Simulations can be run practically on any unix-like environment with a decent and fairly up-to-date C++ compiler, for example gcc 4.x. Certain OMNEST features (Tkenv, parallel simulation, XML support, etc.) depend on the availability of external libraries (Tcl/Tk, MPI, LibXML or Expat, etc.)

IDE platforms are restricted because the IDE relies on a native shared library, which we compile for the above platforms and distribute in binary form for convenience.

Chapter 2. Windows

2.1. Supported Windows Versions

The supported Windows versions are the Intel 32-bit versions of Windows XP, and later versions such as Windows 7 and 8.

64-bit Windows versions are also supported, but be aware that binaries bundled with OMNEST are 32-bit ones, and simulations will also be compiled in 32-bit mode.

2.2. Pre-installation Steps

Download `omnest-4.5-win32.exe` from http://www.omnest.com.

The MinGW GCC compiler is bundled with OMNEST. You do not need to install any additional compiler to work with OMNEST. If you would like to use Microsoft Visual C++ instead of the bundled MinGW GCC compiler, you should install the Microsoft Visual C++ IDE along with the Microsoft Windows SDK before installing OMNEST.

2.3. Installing OMNEST

Find the downloaded installation file using Windows Explorer, and double-click the file. This will start the installation process.

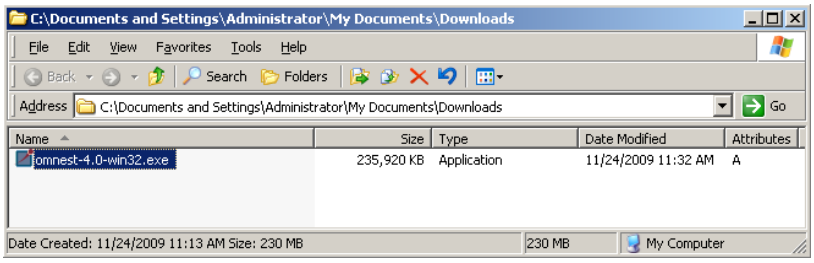

Figure 2.1. Starting the installer

To start the installation, accept the Licensing agreements:

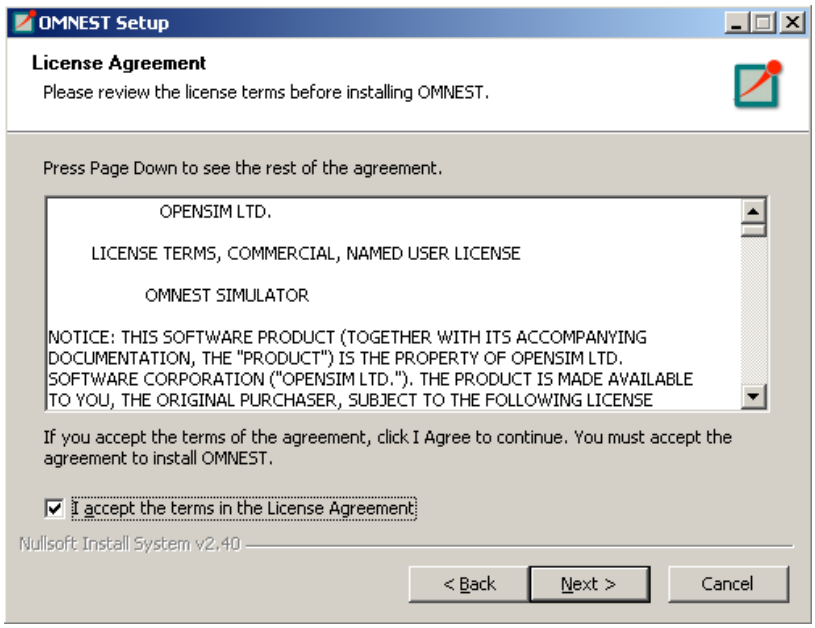

Figure 2.2. License agreement

Select an installation target directory. Please make sure that the installation path does not contain spaces.

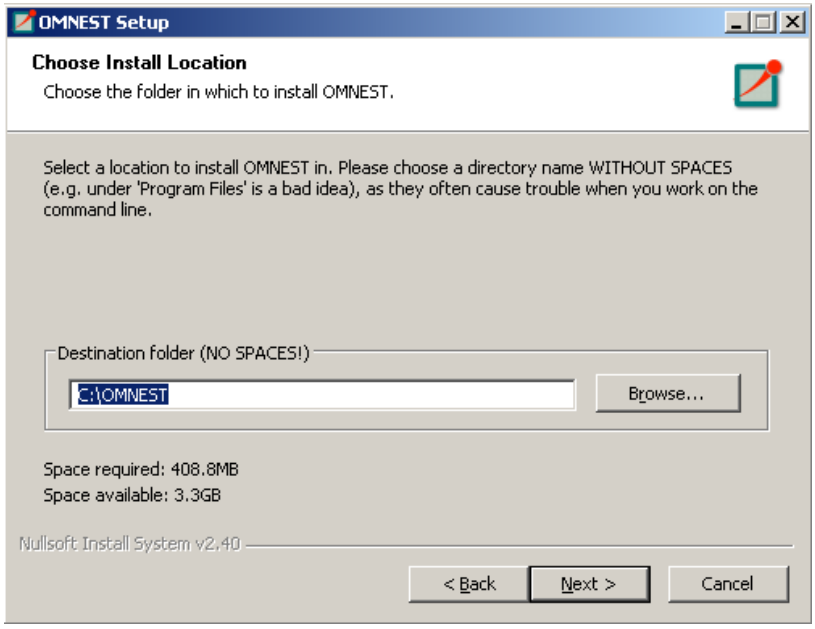

Figure 2.3. Selecting the installation directory

On the next page, select additional components that should be installed along with the core OMNEST files. The installer comes with pre-built shared libraries for different compilers. You can choose to install both MinGW and Visual C++ libraries if you plan to use both compilers. If you do not install any pre-built binary packages, you have to compile OMNEST manually after the installation has finished.

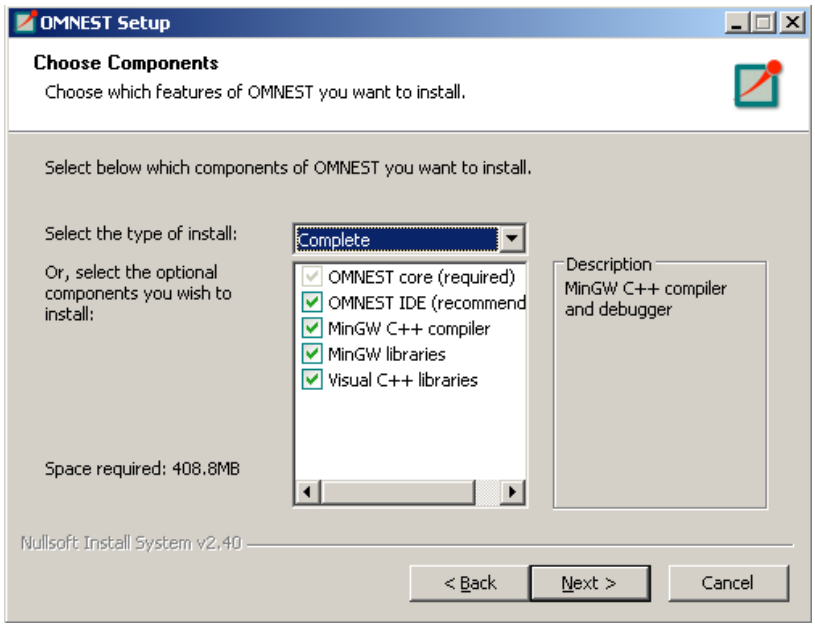

Figure 2.4. Optional components, pre-built libraries

On the next page you have to specify which compiler you intend to use with OMNEST. This can be the bundled MinGW GCC compiler, or Microsoft Visual C++. Microsoft Visual Studio installations (version 11 or 12) detected by the installer will be marked on the list.

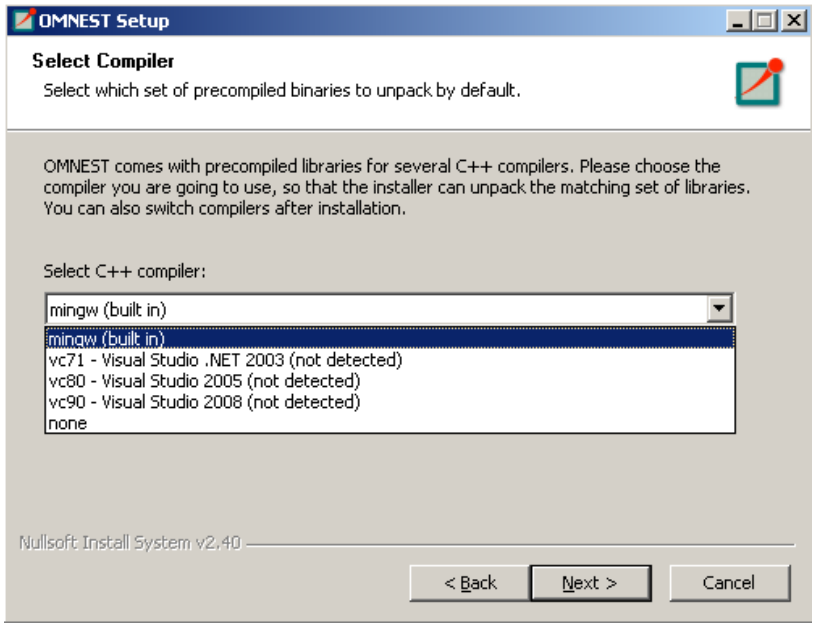

Figure 2.5. Compiler selection

On the last page, you can specify if you want to install the optional WinPcap package. This component is used by the INET Framework to capture network packets on physical network interfaces. Install it only if you plan to do simulations that interact with real networks.

You can also choose if you want to create program launcher icons on your desktop.

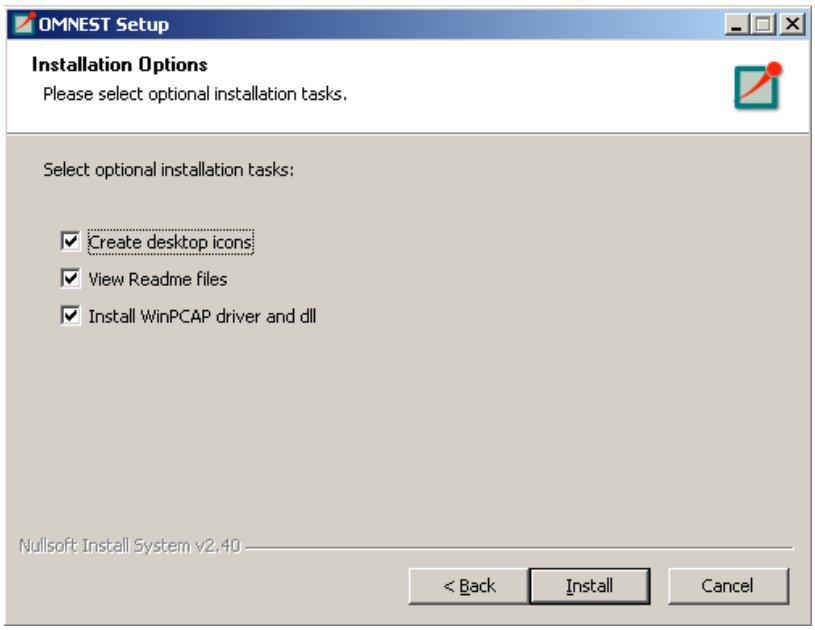

Figure 2.6. Installation options

After this step the installation process starts, and all files required by OMNEST will be copied to the installation folder. At the end of the process, the optional WinPcap installer is launched if you have selected it on the options page.

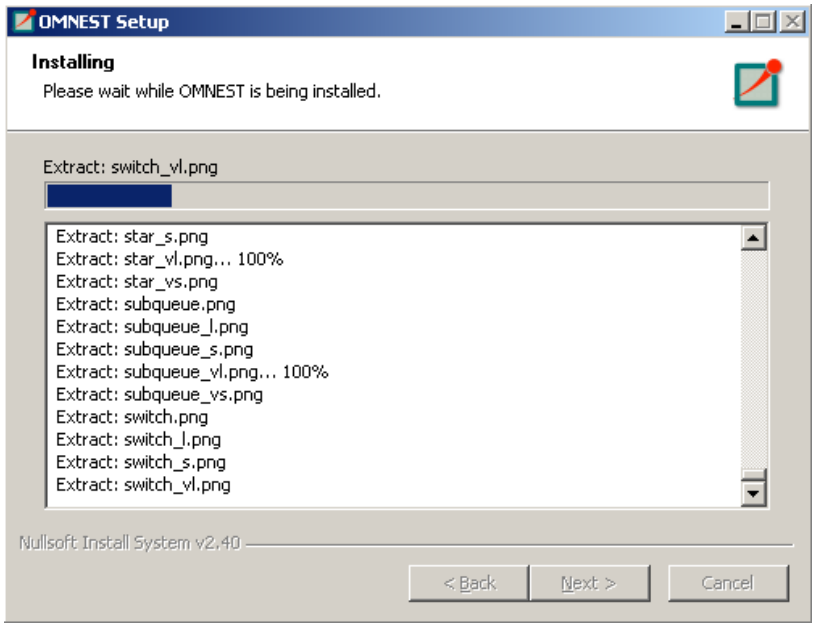

Figure 2.7. Installation options

Finally, a new Start Menu folder is created (along with desktop shortcuts). You can start the *OMNEST Shell* or the *OMNEST IDE* by clicking on the icons.

If you want to work from the command line, use the provided *OMNEST Shell*. This shell sets all environment variables and the path necessary to run OMNEST simulations.

If you have chosen to install OMNEST using the pre-built MS Visual C++ binaries, but you do not have Visual Studio installed yet, chances are that Microsoft Visual C++

Runtime is not installed either. You will receive errors about missing DLL files when you try to run the pre-compiled sample simulations. To install the Visual C++ Runtime package, run the bundled `vcredist_x86_vc11.exe` or `vcredist_x86_vc12.exe` file which you can find in the `<installdir>/store` directory.

2.4. Using the IDE

Once the installation has finished, you can start using OMNEST by launching the IDE. The IDE can be launched with the corresponding Start Menu or desktop shortcut, or by typing `omnest` at the OMNEST Shell prompt.

The simulation examples can be launched from either the IDE, directly with the corresponding Start Menu shortcut, or by changing into the `samples/` directory in the OMNEST Shell and typing `./rundemo`.

In the IDE, each simulation example is a separate project. To build an example, open its project using the context menu (right-click, *Open Project*), and click the *Run* button on the toolbar. To rebuild the example, first make sure that the correct toolchain is selected (context menu, *Build Configurations* > *Set Active*), then choose *Clean Project* and *Build Project* from the context menu.

The IDE is documented in detail in the *User Guide*.

2.5. Using OMNEST with the MinGW GCC Compiler

The bundled MinGW GCC compiler is preconfigured for OMNEST. Note that only the bundled version of MinGW has been tested and is supported with OMNEST.

If you have installed the pre-compiled binary package for the MinGW compiler, make sure that the `<installdir>/bin` and `<installdir>/lib` directories contain the correct executables and libraries. You should see `libopp*.dll` files in the `<installdir>/bin` directory and a `gcc` sub-directory in `<installdir>/lib`. If you do not see them, check the "Switching Compilers" or the "Recompiling OMNEST" section before proceeding.

To test the installation, try to run models from the `<installdir>/lib` directory.

2.5.1. Compiling Simulations on the Command Line

To build simulations from the command line, the following directories have to be included in the path: `<installdir>\bin`, `<installdir>\msys\bin` and `<installdir>\mingw\bin`.

OMNEST provides a *OMNEST MinGW Shell* window (`mingwenv.cmd`), which sets the path and environment variables.

Before compiling a model, you must generate a `Makefile` for it. Change into the model directory and execute:

```
$ opp_makemake -f --deep
```

This command will generate a `Makefile` that can compile all your .cc files in the model directory.

```
$ make
```

will compile and build your project.

Be sure to check the *Manual* for all the options and features of `opp_makemake`.

2.5.2. Recompiling OMNEST

Open the *OMNEST MinGW Shell* window and type:

```
$ ./configure
```

This command will detect all required software on your machine, and configure your build environment. The configuration process creates a file called `Makefile.inc` in your installation root. This file will be included in all of your makefiles, and contains all variables, paths and settings for the build process.

If you do not have binary files in your `bin` directory (no pre-compiled binaries were installed), you should compile OMNEST now manually by typing:

```
$ make
```

The above command creates both debug and release versions of the libraries. If you want to create only one type, use the `make MODE=debug` or `make MODE=release` commands.

If you have a dual-core machine, you can speed up the compilation by adding the `-j2` option to the `make` command line, which enables parallel build support.

2.6. Using OMNEST with Microsoft Visual C++

OMNEST comes with pre-built binaries for the Visual C++ 11 (2012) and 12 (2013) compilers. If you have installed the pre-compiled binary package for the Visual C++ compiler, make sure that the `<installdir>/bin` and `<installdir>/lib` directories contain the correct executables and libraries. You should see `opp*.dll` files in the `<installdir>/bin` directory and a `vc110` or `vc120` sub-directory in `<installdir>/lib`. If you do not see them, check the "Switching Compilers" or the "Recompiling OMNEST" section before proceeding.

To test the installation try to run models from the `<installdir>/lib` directory.

For now, the OMNEST IDE cannot be used for debugging Visual C++ binaries. This is a limitation of the Eclipse CDT component that OMNEST build on. We recommend that you use the Visual Studio IDE for debugging.

2.6.1. Compiling Simulations on the Command Line

To build simulations from the command line, the following directories have to be included in the path: `<installdir>\bin`, `<installdir>\msys\bin` and the directories required by Visual C+. To include the required Visual C+ directories, VC provides a batch file called `vcvars32.bat` in its `bin` directory.

OMNEST provides a *OMNEST Visual C++ Shell* window (`vcenv.cmd`), which sets the path and environment variables.

You may need to adjust your Visual C++ directory in the file.

Before compiling a model, you must generate a `Makefile.vc` for it. Change into the model directory and execute:

```
> opp_nmakemake -f --deep
```

This command will generate a `Makefile.vc` file that can compile all your .cc files in the model directory.

```
> nmake -f Makefile.vc
```

will build your project.

2.6.2. Compiling Simulations from the IDE

When compiling simulations from the IDE, make sure that the correct build configuration (*msvc-debug* or *msvc-release*) is selected.

2.6.3. Recompiling OMNEST

Open the *OMNEST Visual C++ Shell* window and check the contents of the `configuser.vc` file. This file will be included in all of your makefiles, and contains all variables, paths and settings for the build process.

If you do not have binary files in your `bin` directory (no pre-compiled binaries were installed), you should compile OMNEST now manually by typing:

```
> nmake -f Makefile.vc
```

2.7. Switching Compilers

If you want to switch compilers after you have installed OMNEST, first you have to delete all executable files generated by that compiler.

Open the *OMNEST MinGW Shell* window (`mingwenv.cmd`) and clean you project, by executing the following command in `<installdir>`

```
$ make cleanall
```

Pre-built binaries are stored in the `<installdir>/store` directory. You must extract their content in the root OMNEST directory. Execute:

```
$ tar xvfz store/mingw-bin.tgz
```

or

```
$ tar xvfz store/vc110-bin.tgz
```

or

```
$ tar xvfz store/vc120-bin.tgz
```

depending on your compiler. After extracting the executables you will be able to run the sample simulations immediately.

Be sure to modify the path to your Visual C++ installation in the `<installdir>\vcenv.cmd` file if you are switching between different versions of Visual C++.

2.8. Additional Packages

Note that Doxygen and GraphViz are already included in the OMNEST package, and will be used by the IDE automatically.

2.8.1. MPI

MPI is only needed if you would like to run parallel simulations.

There are several MPI implementations for Windows, and OMNEST does not mandate any specific one. We recommend DeinoMPI, which can be downloaded from http://mpi.deino.net.

DeinoMPI ships with binaries compiled with MSVC. After installing DeinoMPI, adjust the `MPI_DIR` setting in `configuser.vc`, and recompile OMNEST with the version of MSVC that matches the DeinoMPI binaries.

In general, if you would like to run parallel simulations, we recommend that you use Linux, OS X, or another unix-like platform.

2.8.2. PCAP

The optional WinPcap library allows simulation models to capture and transmit network packets bypassing the operating system's protocol stack. It is not used directly by OMNEST, but OMNEST detects the necessary compiler and linker options for models in case they need it.

2.8.3. Akaroa

Akaroa 2.6.7, which is the latest version at the time of writing, does not support Windows. You may try to port it using the porting guide from the Akaroa distribution.

2.8.4. SystemC

To enable SystemC integration for Visual C++, set `SYSTEMC=yes` in `configuser.vc`, and rebuild your project. You can find SystemC example simulations in the `samples/systemc-embedding` directory.

SystemC integration is not available when using the MinGW compiler. (The bundled SystemC reference implementation itself is not supported on MinGW.)

Chapter 3. Mac OS X

3.1. Supported Releases

This chapter provides additional information for installing OMNEST on Mac OS X.

The following releases are covered:

- Mac OS X 10.7 (*Lion*)
- Mac OS X 10.8 (*Mountain Lion*)
- Mac OS X 10.9 (*Mavericks*)

3.2. Installing the Prerequisite Packages

- Install the Java Runtime from http://support.apple.com/kb/DL1572 , because OS X does not provide it by default.
- Install the command line developer tools for OS X from http://developer.apple.com/downloads/index.action?=Command%20Line%20Tools (you will need a free Apple Developer Account for the download.)
- Install the quartz project from http://xquartz.macosforge.org/. Quartz will provide some X headers required by the simulation runtime environment.

Installing additional packages will enable more functionality in OMNEST; see the *Additional packages* section at the end of this chapter.

3.3. Additional Steps Required on Mac OS X 10.9 or Later

The Command Line Tools package on Mac OS X 10.9 no longer contains gcc and gdb; instead it contains the Clang compiler and lldb. (The `gcc` and `g++` commands actually run `clang`.) OMNEST will use Clang automatically. However, the OMNEST IDE can only use gdb as the underlying debugger, but not lldb.

To be able to debug from the IDE, you have to install gdb from MacPorts. Alternatively, you can use XCode for debugging.

To do the former, first install MacPorts from http://macports.org. Then you can install gdb:

```
$ sudo port install gdb
```

OS X 10.9 requires that you sign the `ggdb` executable with a self-signed certificate (or with your own certificate, if you have one.)

Start the *Keychain Access* application. Choose *Keychain Access* > *Certificate Assistant* > *Create a Certificate...* from the menu.

Choose a name (e.g. `gdb-cert`), set *Identity Type* to *Self Signed Root*, set *Certificate Type* to *Code Signing* and select the *Let me override defaults*. Click several times on *Continue* until you get to the *Specify a Location For The Certificate* screen, then set *Keychain* to *System*.

If you can't store the certificate in the *System* keychain, create it in the *Login* keychain, then export it. You can then import it into the *System* keychain.

Finally, using the context menu for the certificate, select *Get Info*, open the *Trust* item, and set *Code Signing* to *Always Trust*.

You must quit the *Keychain Access* application in order to use the certificate and restart the system.

Now sign the executable:

```
$ sudo codesign -s gdb-cert /opt/local/bin/ggdb
```

After installing the OMNEST IDE, you have to change the name of the executable used for debugging. Go to *Preferences* > *C++* > *Debug* > *gdb* in the IDE, and change the executable name from `gdb` to `ggdb`.

3.4. Downloading and Unpacking OMNEST

Download OMNEST from http://omnest.com. Make sure you select to download the generic archive, `omnest-4.5-src.tgz`.

Copy the archive to the directory where you want to install it. This is usually your home directory, `/Users/<you>`. Open a terminal, and extract the archive using the following command:

```
$ tar zxvf omnest-4.5-src.tgz
```

A subdirectory called `omnest-4.5` will be created, containing the simulator files.

Alternatively, you can also unpack the archive using Finder.

The Terminal can be found in the Applications / Utilities folder.

3.5. Environment Variables

OMNEST needs its `bin/` directory to be in the path. To add `bin/` to `PATH` temporarily (in the current shell only), change into the OMNEST directory and source the `setenv` script:

```
$ cd omnest-4.5
$ . setenv
```

To set the environment variables permanently, edit `.bashrc` in your home directory. Use your favourite text editor to edit `.bashrc`, for example TextEdit:

```
$ touch ~/.bashrc
$ open -e ~/.bashrc
```

`touch` is needed because `open -e` only opens existing files. Alternatively, you can use the terminal-based *pico* editor (`pico ~/.bashrc`)

Add the following line at the end of the file, then save it:

```
export PATH=$PATH:$HOME/omnest-4.5/bin
```

You need to close and re-open the terminal for the changes to take effect.

Alternatively, you can put the above line into `~/.bash_profile`, but then you need to log out and log in again for the changes to take effect.

If you use a shell other than the default one, *bash*, consult the man page of that shell to find out which startup file to edit, and how to set and export variables.

3.6. Configuring and Building OMNEST

Check `configure.user` to make sure it contains the settings you need. In most cases you don't need to change anything in it.

In the top-level OMNEST directory, type:

```
$ ./configure
```

The `configure` script detects installed software and configuration of your system. It writes the results into the `Makefile.inc` file, which will be read by the makefiles during the build process.

Normally, the `configure` script needs to be running under the graphical environment in order to test for `wish`, the Tcl/Tk shell. If you are logged in via an ssh session or you want to compile OMNEST without Tcl/Tk, use the command

```
$ NO_TCL=1 ./configure
```

instead of plain `./configure`.

If there is an error during `configure`, the output may give hints about what went wrong. Scroll up to see the messages. (You may need to increase the scrollback buffer size of the terminal and re-run `./configure`.) The script also writes a very detailed log of its operation into `config.log` to help track down errors. Since `config.log` is very long, it is recommended that you open it in an editor and search for phrases like *error* or the name of the package associated with the problem.

When `./configure` has finished, you can compile OMNEST. Type in the terminal:

```
$ make
```

To take advantage of multiple processor cores, add the `-j4` option to the `make` command line.

The build process will not write anything outside its directory, so no special privileges are needed.

The make command will seemingly compile everything twice. This is because both debug and optimized versions of the libraries are built. If you only want to build one set of the libraries, specify `MODE=debug` or `MODE=release`:

```
$ make MODE=release
```

3.7. Verifying the Installation

You can now verify that the sample simulations run correctly. For example, the dyna simulation is started by entering the following commands:

```
$ cd samples/dyna
$ ./dyna
```

By default, the samples will run using the Tcl/Tk environment. You should see nice gui windows and dialogs.

3.8. Starting the IDE

OMNEST comes with an Eclipse-based Simulation IDE. On Mac OS X 10.7 (Lion) or later, the Java Runtime must be installed (see prerequisites) before you can use the IDE. Start the IDE by typing:

```
$ omnest
```

If you would like to be able to launch the IDE via Applications, the Dock or a desktop shortcut, do the following: open the `omnest-4.5` folder in Finder, go into the `ide` subfolder, create an alias for the omnest program there (right-click, *Make Alias*), and drag the new alias into the Applications folder, onto the Dock, or onto the desktop.

Alternatively, run one or both of the commands below:

```
$ make install-menu-item
$ make install-desktop-icon
```

which will do roughly the same.

3.9. Using the IDE

When you try to build a project in the IDE, you may get the following warning message:

> Toolchain "..." is not supported on this platform or installation. Please go to the Project menu, and activate a different build configuration. (You may need to switch to the C/C++ perspective first, so that the required menu items appear in the Project menu.)

If you encounter this message, choose *Project > Properties > C/C++ Build > Tool Chain Editor > Current toolchain > GCC for OMNEST*.

The IDE is documented in detail in the *User Guide*.

3.10. Reconfiguring the Libraries

If you need to recompile the OMNEST components with different flags (e.g. different optimization), then change the top-level OMNEST directory, edit `configure.user` accordingly, then type:

```
$ ./configure
$ make clean
$ make
```

To take advantage of multiple processor cores, add the `-j2` option to the `make` command line.

If you want to recompile just a single library, then change to the directory of the library (e.g. `cd src/sim`) and type:

```
$ make clean
$ make
```

By default, libraries are compiled in both debug and release mode. If you want to make release or debug builds only, use:

```
$ make MODE=release
```

or

```
$ make MODE=debug
```

By default, shared libraries will be created. If you want to build static libraries, set `SHARED_LIBS=no` in `configure.user` and re-configure your project.

The built libraries and programs are immediately copied to the `lib/` and `bin/` subdirectories.

The Tcl/Tk environment uses the native Aqua version of Tcl/Tk, so you will see native widgets. However, due to problems in the Tk/Aqua port, you may experience minor UI quirks. We are aware of these problems, and are working on the solution.

3.11. Additional Packages

3.11.1. OpenMPI

OS X does not come with OpenMPI, so you must install it manually. Download it from http://open-mpi.org and follow the installation instructions. Alternatively, you can install it from the MacPorts repo by typing `sudo port install openmpi`. In this case, you have to manually set the MPI_CFLAGS and MPI_LIBS variables in `configure.user` and re-run `./configure`. Please note that we do not provide support for OpenMPI installed from the MacPorts repository.

MacPorts is a repository of several open source packages for Mac OS X. Using MacPorts packages may save you some manual work. You can install MacPorts from http://www.macports.org.

3.11.2. GraphViz

GraphViz is needed if you want to have diagrams in HTML documentation that you generate from NED files in the IDE (*Generate NED Documentation...* item in the project context menu).

Download and install the GraphViz OS X binaries from http://www.pixelglow.com/graphviz/download/. Download the latest, version 2.x package; at the time of writing, the link is at the top of the page.

Alternatively, you can install it from the MacPorts project by typing `sudo port install graphviz`.

After installation, make sure that the `dot` program is available from the command line. Open a terminal, and type

```
$ dot -V
```

Note the capital *V*. The command should normally work out of the box. If you get the "*command not found*" error, you need to put `dot` into the path. Find the `dot` program in the GraphViz installation directory, and soft link it into `/usr/local/bin` (`sudo ln -s <path>/dot /usr/local/bin`).

3.11.3. Doxygen

Doxygen is needed if you want to generate documentation for C++ code, as part of the HTML documentation that you generate from NED files in the IDE (*Generate NED Documentation...* item in the project context menu).

Download the Doxygen OS X binaries from the Doxygen web site's download page, http://www.stack.nl/~dimitri/doxygen/download.html, and install it.

Alternatively, you can install it from the MacPorts project by typing `sudo port install doxygen`.

After installation, ensure that the `doxygen` program is available from the command line. Open a terminal, and type

```
$ doxygen
```

If you get the "*command not found*" error, you need to put `doxygen` into the path. Enter into a terminal:

```
$ cd /Applications/Doxygen.app/Contents/Resources/
$ sudo ln -s doxygen doxytags /usr/local/bin
```

3.11.4. Akaroa

Akaroa 2.7.9, which is the latest version at the time of writing, does not support Mac OS X. You may try to port it using the porting guide from the Akaroa distribution.

3.11.5. SystemC

SystemC integration is not available on Mac OS X, because the bundled SystemC reference implementation does not support Mac OS X 10.7.

Chapter 4. Linux

4.1. Supported Linux Distributions

This chapter provides instructions for installing OMNEST on selected Linux distributions:

- Ubuntu 12.04 LTS, 13.04
- Fedora Core 18
- Red Hat Enterprise Linux Desktop Workstation 6.4
- OpenSUSE 12.3

This chapter describes the overall process. Distro-specific information, such as how to install the prerequisite packages, are covered by distro-specific chapters.

If your Linux distribution is not listed above, you still may be able to use some distro-specific instructions in this Guide.

Ubuntu derivatives (Ubuntu instructions may apply):

- Kubuntu, Xubuntu, Edubuntu, ...
- Linux Mint

Some Debian-based distros (Ubuntu instructions may apply, as Ubuntu itself is based on Debian):

- Knoppix and derivatives
- Mepis

Some Fedora-based distros (Fedora instructions may apply):

- Simplis
- Eeedora

4.2. Installing the Prerequisite Packages

OMNEST requires several packages to be installed on the computer. These packages include the C++ compiler (gcc), the Java runtime, and several other libraries and programs. These packages can be installed from the software repositories of your Linux distribution.

See the chapter specific to your Linux distribution for instructions on installing the packages needed by OMNEST.

You may need superuser permissions to install packages.

Not all packages are available from software repositories; some (optional) ones need to be downloaded separately from their web sites, and installed manually. See the section *Additional Packages* later in this chapter.

4.3. Downloading and Unpacking

Download OMNEST from http://omnest.com. Make sure you select to download the generic archive, `omnest-4.5-src.tgz`.

Copy the archive to the directory where you want to install it. This is usually your home directory, `/home/<you>`. Open a terminal, and extract the archive using the following command:

```
$ tar xvfz omnest-4.5-src.tgz
```

This will create an `omnest-4.5` subdirectory with the OMNEST files in it.

On how to open a terminal on your Linux installation, see the chapter specific to your Linux distribution.

4.4. Environment Variables

OMNEST needs its `bin/` directory to be in the path. To add `bin/` to `PATH` temporarily (in the current shell only), change into the OMNEST directory and source the `setenv` script:

```
$ cd omnest-4.5
$ . setenv
```

The script also adds the `lib/` subdirectory to `LD_LIBRARY_PATH`, which may be necessary on systems that don't support the rpath mechanism.

To set the environment variables permanently, edit `.bashrc` in your home directory. Use your favourite text editor to edit `.bashrc`, for example gedit:

```
$ gedit ~/.bashrc
```

Add the following line at the end of the file, then save it:

```
export PATH=$PATH:$HOME/omnest-4.5/bin
```

You need to close and re-open the terminal for the changes to take effect.

Alternatively, you can put the above line into `~/.bash_profile`, but then you need to log out and log in again for the changes to take effect.

If you use a shell other than *bash*, consult the man page of that shell to find out which startup file to edit, and how to set and export variables.

Note that all Linux distributions covered in this Installation Guide use *bash* unless the user has explicitly selected another shell.

4.5. Configuring and Building OMNEST

In the top-level OMNEST directory, type:

```
$ ./configure
```

The `configure` script detects installed software and configuration of your system. It writes the results into the `Makefile.inc` file, which will be read by the makefiles during the build process.

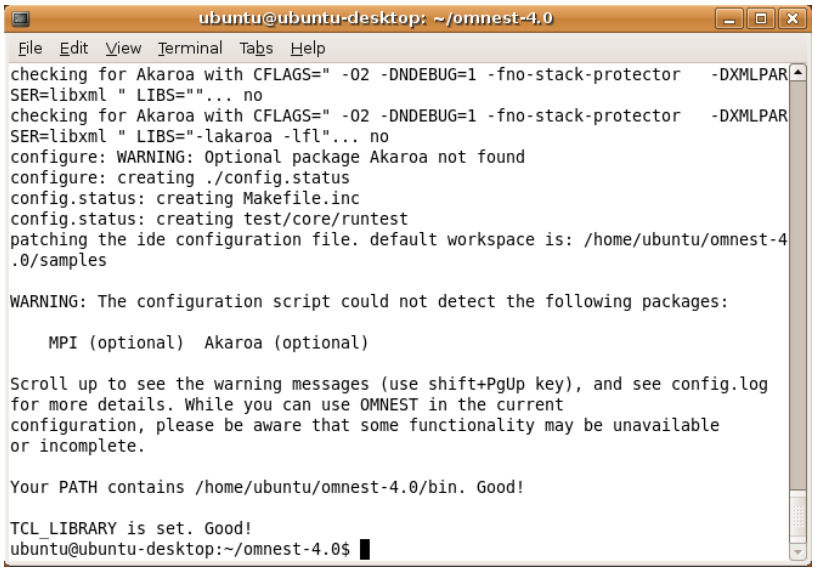

Figure 4.1. Configuring OMNEST

Normally, the `configure` script needs to be running under the graphical environment (X11) in order to test for `wish`, the Tcl/Tk shell. If you are logged in via an *ssh* session, or there is some other reason why X is not running, the easiest way to work around the problem is to tell OMNEST to build without Tcl/Tk. To do that, use the command

```
$ NO_TCL=1 ./configure
```

instead of plain `./configure`.

If there is an error during `configure`, the output may give hints about what went wrong. Scroll up to see the messages. (Use Shift+PgUp; you may need to increase the scrollback buffer size of the terminal and re-run `./configure`.) The script also writes a very detailed log of its operation into `config.log` to help track down errors. Since `config.log` is very long, it is recommended that you open it in an editor and search for phrases like *error* or the name of the package associated with the problem.

When `./configure` has finished, you can compile OMNEST. Type in the terminal:

```
$ make
```

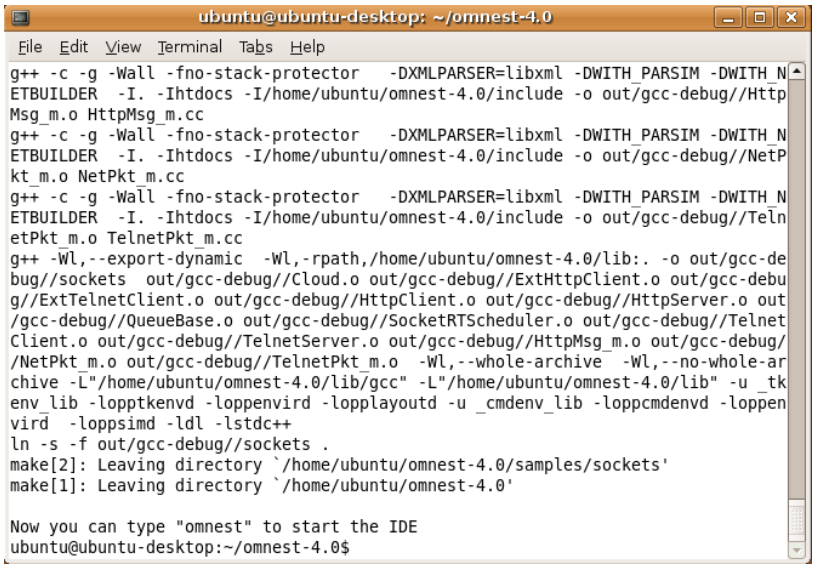

Figure 4.2. Building OMNEST

To take advantage of multiple processor cores, add the `-j2` option to the `make` command line.

The build process will not write anything outside its directory, so no special privileges are needed.

The make command will seemingly compile everything twice. This is because both debug and optimized versions of the libraries are built. If you only want to build one set of the libraries, specify `MODE=debug` or `MODE=release`:

```
$ make MODE=release
```

4.6. Verifying the Installation

You can now verify that the sample simulations run correctly. For example, the dyna simulation is started by entering the following commands:

```
$ cd samples/dyna
$ ./dyna
```

By default, the samples will run using the Tcl/Tk environment. You should see nice gui windows and dialogs.

4.7. Starting the IDE

You can launch the OMNEST Simulation IDE by typing the following command in the terminal:

```
$ omnest
```

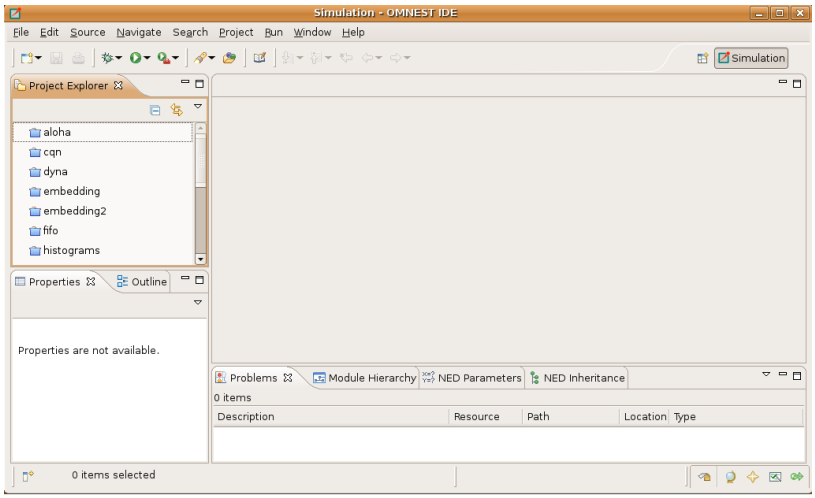

Figure 4.3. The Simulation IDE

If you would like to be able to access the IDE from the application launcher or via a desktop shortcut, run one or both of the commands below:

```
$ make install-menu-item
$ make install-desktop-icon
```

Or add a shortcut that points to the `omnest` program in the `ide` subdirectory by other means, for example using the Linux desktop's context menu.

4.8. Using the IDE

When you try to build a project in the IDE, you may get the following warning message:

> Toolchain "..." is not supported on this platform or installation. Please go to the Project menu, and activate a different build configuration. (You may need to switch to the C/C++ perspective first, so that the required menu items appear in the Project menu.)

If you encounter this message, choose *Project > Properties > C/C++ Build > Tool Chain Editor > Current toolchain > GCC for OMNEST*.

The IDE is documented in detail in the *User Guide*.

4.9. Reconfiguring the Libraries

If you need to recompile the OMNEST components with different flags (e.g. different optimization), then change the top-level OMNEST directory, edit `configure.user` accordingly, then type:

```
$ ./configure
$ make cleanall
$ make
```

If you want to recompile just a single library, then change to the directory of the library (e.g. `cd src/sim`) and type:

```
$ make clean
$ make
```

By default, libraries are compiled in both debug and release mode. If you want to make release or debug builds only, use:

```
$ make MODE=release
```

or

```
$ make MODE=debug
```

By default, shared libraries will be created. If you want to build static libraries, set `SHARED_LIBS=no` in `configure.user` and re-configure your project.

For detailed description of all options please read the *Build Options* chapter.

4.10. Additional Packages

Note that at this point, MPI, Doxygen and GraphViz have been installed as part of the prerequisites.

4.10.1. Akaroa

Linux distributions do not contain the Akaroa package. It must be downloaded, compiled and installed manually before installing OMNEST.

As of version 2.7.9, Akaroa only supports Linux and Solaris.

Download Akaroa 2.7.9 from: http://www.cosc.canterbury.ac.nz/research/RG/net_sim/simulation_group/akaroa/download.chtml

Extract it into a temporary directory:

```
$ tar xfz akaroa-2.7.9.tar.gz
```

Configure, build and install the Akaroa library. By default, it will be installed into the `/usr/local/akaroa` directory.

```
$ ./configure
$ make
$ sudo make install
```

Go to the OMNEST directory, and (re-)run the `configure` script. Akaroa will be automatically detected if you installed it to the default location.

4.10.2. PCAP

The optional Pcap library allows simulation models to capture and transmit network packets bypassing the operating system's protocol stack. It is not used directly by OMNEST, but OMNEST detects the necessary compiler and linker options for models in case they need it.

4.10.3. SystemC

To enable SystemC integration, add SYSTEMC=yes to the configure.user file, run *configure* and then rebuild your project. You can check the systemc examples in the samples/systemc-embedding directory.

Chapter 5. Ubuntu

5.1. Supported Releases

This chapter provides additional information for installing OMNEST on Ubuntu Linux installations. The overall installation procedure is described in the *Linux* chapter.

The following Ubuntu releases are covered:

- Ubuntu 12.04 LTS
- Ubuntu 13.10

They were tested on the following architectures:

- Intel 32-bit and 64-bit

The instructions below assume that you use the Gnome desktop and the bash shell, which are the defaults. If you use another desktop environment or shell, you may need to adjust the instructions accordingly.

5.2. Opening a Terminal

Type *terminal* in Dash and click on the Terminal icon.

5.3. Installing the Prerequisite Packages

You can perform the installation using the graphical user interface or from the terminal, whichever you prefer.

5.3.1. Command-Line Installation

Before starting the installation, refresh the database of available packages. Type in the terminal:

```
$ sudo apt-get update
```

To install the required packages, type in the terminal:

```
$ sudo apt-get install build-essential gcc g++ bison flex perl \
      tcl-dev tk-dev libxml2-dev zlib1g-dev default-jre \
      doxygen graphviz libwebkitgtk-1.0-0 openmpi-bin \
      libopenmpi-dev libpcap-dev
```

At the confirmation questions (*Do you want to continue? [Y/N]*), answer *Y*.

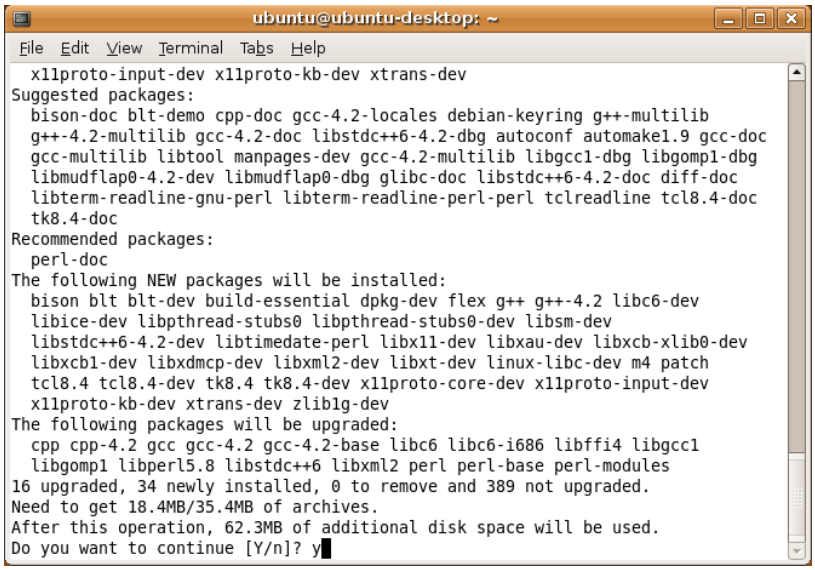

Figure 5.1. Command-Line Package Installation

5.3.2. Graphical Installation

Ubuntu's graphical installer, *Synaptic*, can be started with the *System* > *Administration* > *Synaptic package manager* menu item.

Since software installation requires root permissions, Synaptic will ask you to type your password.

Search for the following packages in the list, click the squares before the names, then choose *Mark for installation* or *Mark for upgrade*.

If the *Mark additional required changes?* dialog comes up, choose the *Mark* button.

The packages:

- build-essential, gcc, g++, bison, flex, perl, tcl-dev, tk-dev, libxml2-dev, zlib1g-dev, default-jre, doxygen, graphviz, libwebkitgtk-1.0-0, openmpi-bin, libopenmpi-dev, libpcap-dev

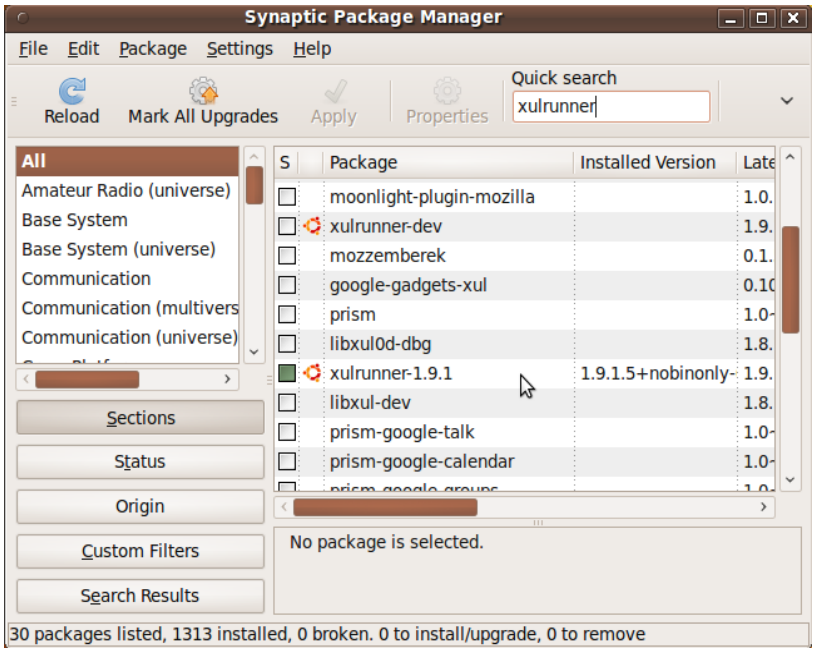

Figure 5.2. Synaptic Package Manager

Click *Apply*, then in the *Apply the following changes?* window, click *Apply* again. In the *Changes applied* window, click *Close*.

5.3.3. Post-Installation Steps

The default tooltip background color in Ubuntu is black, which causes certain tooltips in the OMNEST IDE to become unreadable (black-on-black). This annoyance can be resolved by changing the tooltip colors in Ubuntu.

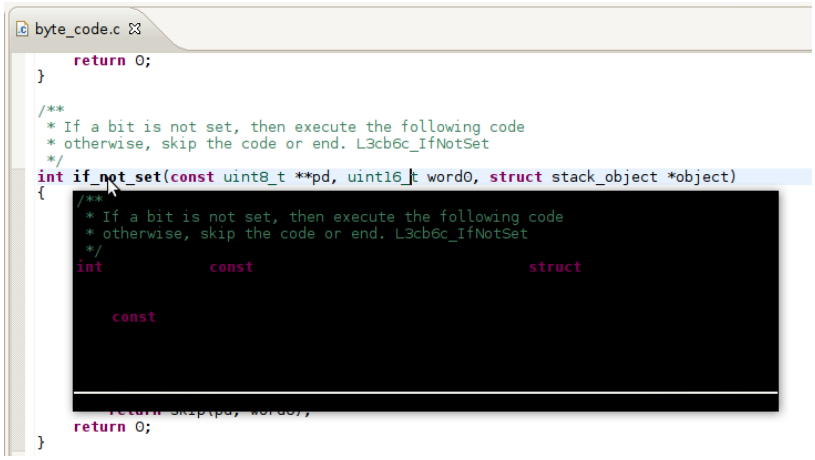

Figure 5.3. Black-on-black text in tooltips

Install *gnome-color-chooser*:

```
$ sudo apt-get install gnome-color-chooser
```

Run it:

```
$ gnome-color-chooser
```

Find the *Tooltips* group on the *Specific* tab, and change the settings to black foreground over pale yellow background. Click Apply.

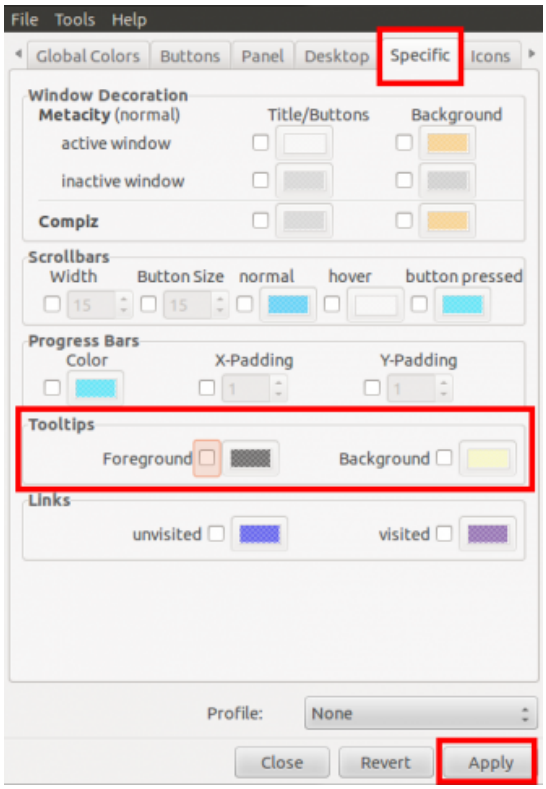

Figure 5.4. Fixing the tooltip color issue

Chapter 6. Fedora 18

6.1. Supported Releases

This chapter provides additional information for installing OMNEST on Fedora installations. The overall installation procedure is described in the *Linux* chapter.

The following Fedora release is covered:

- Fedora 18

It was tested on the following architectures:

- Intel 32-bit and 64-bit

6.2. Opening a Terminal

Choose *Applications* > *System Tools* > *Terminal* from the menu.

6.3. Installing the Prerequisite Packages

You can perform the installation using the graphical user interface or from the terminal, whichever you prefer.

6.3.1. Command-Line Installation

To install the required packages, type in the terminal:

```
$ su -c 'yum install make gcc gcc-c++ bison flex perl \
    tcl-devel tk-devel libxml2-devel zlib-devel \
    java doxygen graphviz webkitgtk openmpi-devel libpcap-devel'
```

then follow the instruction on the console.

Note that *openmpi* will not be available by default, it needs to be activated in every session with the

```
$ module load openmpi-<arch>
```

command, where `<arch>` is your architecture (usually `i386` or `x86_64`). When in doubt, use `module avail` to display the list of available modules. If you need MPI in every session, you may add the `module load` command to your startup script (`.bashrc`).

6.3.2. Graphical Installation

The graphical installer can be launched by choosing *System* > *Administration* > *Add/Remove Software* from the menu.

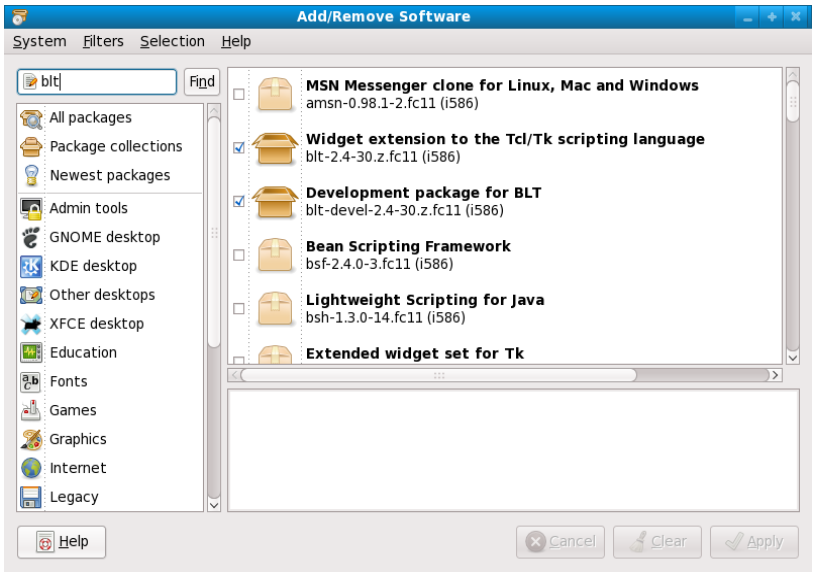

Figure 6.1. Add/Remove Software

Search for the following packages in the list. Select the checkboxes in front of the names, and pick the latest version of each package.

The packages:

- bison, gcc, gcc-c++, flex, perl, tcl-devel, tk-devel, libxml2-devel, zlib-devel, webkitgtk, make, java, doxygen, graphviz, openmpi-devel, libpcap-devel

Click *Apply*, then follow the instructions.

Chapter 7. Red Hat

7.1. Supported Releases

This chapter provides additional information for installing OMNEST on Red Hat Enterprise Linux installations. The overall installation procedure is described in the *Linux* chapter.

The following Red Hat release is covered:

- Red Hat Enterprise Linux Desktop Workstation 6.4

It was tested on the following architectures:

- Intel 32-bit and 64-bit

7.2. Opening a Terminal

Choose *Applications* > *Accessories* > *Terminal* from the menu.

7.3. Installing the Prerequisite Packages

You can perform the installation using the graphical user interface or from the terminal, whichever you prefer.

You will need Red Hat Enterprise Linux Desktop Workstation for OMNEST. The *Desktop Client* version does not contain development tools.

7.3.1. Command-Line Installation

To install the required packages, type in the terminal:

```
$ su -c 'yum install make devtoolset-1.1 bison flex perl \
    tcl-devel tk-devel libxml2-devel zlib-devel \
    java doxygen graphviz openmpi-devel libpcap'
```

After installing the *devtoolset-1.1* package you must make it available to OMNEST:

```
$ scl enable devtoolset-1.1 'bash'
```

This command will open a new shell configured to use the newest available version of the GCC toolchain. Follow the general Linux installation instructions using this shell.

Activate the devtoolset package in the terminal each time when you work with OMNEST and start the IDE from that shell.

RedHat contains two separate GCC toolchains. The *devtoolset* package contains the latest available version of GCC while the system default gcc and gcc-c++ packages are using a much older version. You may opt to use the system default GCC toolchain by installing the gcc and gcc-c++ packages instead of *devtools-1.1*. In this case you don't have to use the above *scl* command.

To install additional (optional) packages for parallel simulation and packet capture support, type:

```
$ su -c 'yum install openmpi-devel libpcap'
```

Note that *openmpi* will not be available by default, it needs to be activated in every session with the

```
$ module load openmpi_<arch>
```

command, where `<arch>` is your architecture (usually `i386` or `x86_64`). When in doubt, use `module avail` to display the list of available modules. If you need MPI in every session, you may add the `module load` command to your startup script (`.bashrc`).`

7.3.2. Graphical Installation

The graphical installer can be launched by choosing *Applications* > *Add/Remove Software* from the menu.

Search for the following packages in the list. Select the checkboxes in front of the names, and pick the latest version of each package.

The packages:

- devtoolset-1.1, bison, flex, perl, tcl-devel, tk-devel, libxml2-devel, zlib-devel, make, java, doxygen, graphviz, openmpi-devel, libpcap

Click *Apply*, then follow the instructions.

7.4. SELinux

You may need to turn off SELinux when running certain simulations. To do so, click on *System* > *Administration* > *Security Level* > *Firewall*, go to the *SELinux* tab, and choose *Disabled*.

You can verify the SELinux status by typing the `sestatus` command in a terminal.

From OMNEST 4.1 on, makefiles that build shared libraries include the `chcon -t textrel_shlib_t lib<name>.so` command that properly sets the security context for the library. This should prevent the SELinux-related "*cannot restore segment prot after reloc: Permission denied*" error from occurring, unless you have a shared library which was built using an obsolete or hand-crafted makefile that does not contain the `chcon` command.

Chapter 8. OpenSUSE

8.1. Supported Releases

This chapter provides additional information for installing OMNEST on openSUSE installations. The overall installation procedure is described in the *Linux* chapter.

The following openSUSE release is covered:

- openSUSE 12.3

It was tested on the following architectures:

- Intel 32-bit and 64-bit

8.2. Opening a Terminal

Choose *Applications* > *System* > *Terminal* > *Terminal* from the menu.

8.3. Installing the Prerequisite Packages

You can perform the installation using the graphical user interface or from the terminal, whichever you prefer.

8.3.1. Command-Line Installation

To install the required packages, type in the terminal:

```
$ sudo zypper install make gcc gcc-c++ bison flex perl \
    tcl-devel tk-devel libxml2-devel zlib-devel \
    java-1_7_0-openjdk doxygen graphviz openmpi-devel libpcap-devel \
    libwebkitgtk-1_0-0
```

then follow the instruction on the console.

Note that *openmpi* will not be available by default, first you need to log out and log in again, or source your `.profile` script:

```
$ . ~/.profile
```

8.3.2. Graphical Installation

The graphical installer can be launched by choosing *Computer* > *Yast* > *System* > *Software* > *Software Management* from the menu.

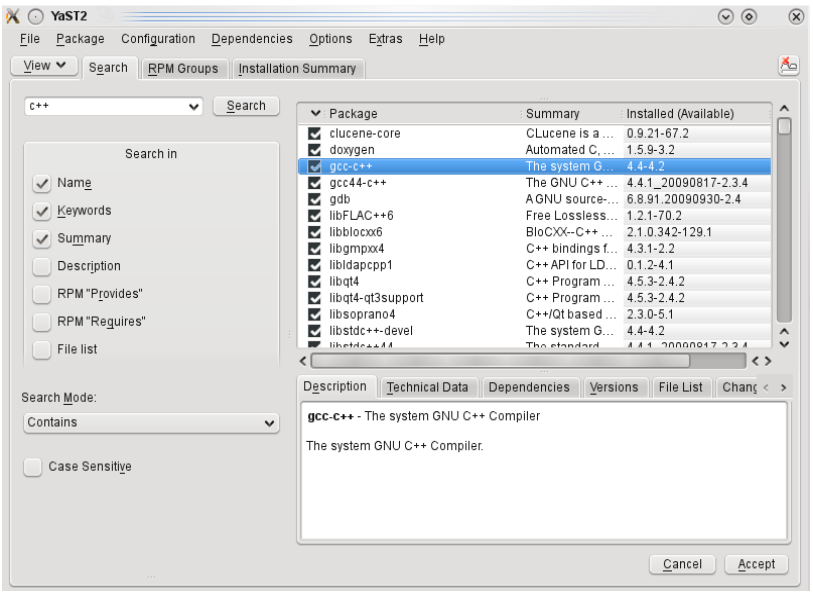

Figure 8.1. Yast Software Management

Search for the following packages in the list. Select the checkboxes in front of the names, and pick the latest version of each package.

The packages:

- make, gcc, gcc-c++, bison, flex, perl, tcl-devel, tk-devel, libxml2-devel, zlib-devel, java-1_7_0-openjdk, doxygen, graphviz, libwebkitgtk-1_0-0, openmpi-devel, libpcap-devel

Click *Accept*, then follow the instructions.

Chapter 9. Generic Unix

9.1. Introduction

This chapter provides additional information for installing OMNEST on Unix-like operating systems not specifically covered by this Installation Guide. The list includes FreeBSD, Solaris, and Linux distributions not covered in other chapters.

In addition to Windows and Mac OS X, the Simulation IDE will only work on Linux x86 32/64-bit platforms. Other operating systems (FreeBSD, Solaris, etc.) and architectures may still be used as simulation platforms, without the IDE.

9.2. Dependencies

The following packages are required for OMNEST to work:

build-essential, GNU make, gcc, g++, bison (2.x+), flex, perl	These packages are needed for compiling OMNEST and simulation models, and also for certain OMNEST tools to work. Some C++ compilers other than g++, for example the Intel compiler, will also be accepted.

The following packages are strongly recommended, because their absence results in severe feature loss:

Tcl/Tk 8.5 or later	Required by the Tkenv simulation runtime environment. You need the *devel* packages that include the C header files as well. It is also possible to compile OMNEST without Tcl/Tk (and Tkenv), by turning on the `NO_TCL` environment variable.
LibXML2 or Expat	Either one of these XML parsers are needed for OMNEST to be able to read XML files. The *devel* packages (that include the header files) are needed. LibXML2 is the preferred one.
SUN JRE or OpenJDK, version 7.0 or later	The Java runtime is required to run the Eclipse-based Simulation IDE. Other implementations, for example Kaffe, have been found to have problems running the IDE. You do not need this package if you do not plan to use the Simulation IDE.
xulrunner	This package is part of Firefox, and is needed by the IDE to display documentation, styled tooltips and other items.

The following packages are required if you want to take advantage of some advanced OMNEST features:

GraphViz, Doxygen	These packages are used by the NED documentation generation feature of the IDE. When they are missing, documentation will have less content.
MPI	openmpi or some other MPI implementation is required to support parallel simulation execution.

Akaroa	Implements Multiple Replications In Parallel (MRIP). Akaroa can be downloaded from the project's website.
Pcap	Allows simulation models to capture and transmit network packets bypassing the operating system's protocol stack. It is not used directly by OMNEST, but OMNEST detects the necessary compiler and linker options for models in case they need it.

The exact names of these packages may differ across distributions.

9.3. Determining Package Names

If you have a distro unrelated to the ones covered in this Installation Guide, you need to figure out what is the established way of installing packages on your system, and what are the names of the packages you need.

9.3.1. Tcl/Tk

Tcl/Tk may be present as separate packages (`tcl` and `tk`), or in one package (`tcltk`). The version number (e.g. 8.5) is usually part of the name in some form (`85`, `8.5`, etc). You will need the development packages, which are usually denoted with the `-dev` or `-devel` name suffix.

Troubleshooting:

If your platform does not have suitable Tcl/Tk packages, you may still use OMNEST to run simulations from the command line. To disable the graphical runtime environment use:

```
$ NO_TCL=yes ./configure
```

This will prevent the build system to link with Tcl/Tk libraries. This is required also if you are installing OMNEST from a remote terminal session.

By default, the `configure` script expects to find the Tcl/Tk libraries in the standard linker path (without any `-Ldirectory` linker option) and under the standard names (i.e. with the `-ltcl8.4` or `-ltcl84` linker option). If you have them in different places or under different names, you have to edit `configure.user` and explicitly set `TK_LIBS` there (see the Build Options chapter for further details).

If you get the error *no display and DISPLAY environment variable not set*, then you're either not running X (the `wish` command, and thus `./configure` won't work just in the console) or you really need to set the `DISPLAY` variable (`export DISPLAY=:0.0` usually does it).

If you get the error: *Tcl_Init failed: Can't find a usable init.tcl...*

The `TCL_LIBRARY` environment variable should point to the directory which contains `init.tcl`. That is, you probably want to put a line like

```
export TCL_LIBRARY=/usr/lib/tcl8.4
```

into your `~/.bashrc`.

If you still have problems installing Tcl/Tk, we recommend visiting the OMNEST site's wiki packages for further troubleshooting tips: http://www.omnetpp.org/pmwiki/index.php?n=Main.TclTkRelatedProblems

9.3.2. The Java Runtime

You need to install the Oracle JRE or OpenJDK. The IDE is not supported on Unix platforms other than Linux, so JRE is not required either. We have tested various other Java runtimes (*gcj*, *kaffe*, etc.), and the IDE does not work well with them.

Java version 6.0 (i.e. JRE 1.6) or later is required and 7.0 is recommended.

9.3.3. MPI

OMNEST is not sensitive to the particular MPI implementation. You may use OpenMPI, or any other standards-compliant MPI package.

9.4. Downloading and Unpacking

Download OMNEST from http://omnest.com. Make sure you select to download the generic archive, `omnest-4.5-src.tgz`.

Copy the archive to the directory where you want to install it. This is usually your home directory, `/home/<you>`. Open a terminal, and extract the archive using the following command:

```
$ tar xvfz omnest-4.5-src.tgz
```

This will create an `omnest-4.5` subdirectory with the OMNEST files in it.

9.5. Environment Variables

In general OMNEST requires that its bin directory should be in the PATH. You should add a line something like this to your `.bashrc`:

```
$ export PATH=$PATH:$HOME/omnest-4.5/bin
```

You may also have to specify the path where shared libraries are loaded from. Use:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/omnest-4.5/lib
```

If `configure` complains about not finding the Tcl library directory, you may specify it by setting the `TCL_LIBRARY` environment variable.

If you use a shell other than bash, consult the man page of that shell to find out which startup file to edit, and how to set and export variables.

9.6. Configuring and Building OMNEST

In the top-level OMNEST directory, type:

```
$ ./configure
```

The `configure` script detects installed software and configuration of your system. It writes the results into the `Makefile.inc` file, which will be read by the makefiles during the build process.

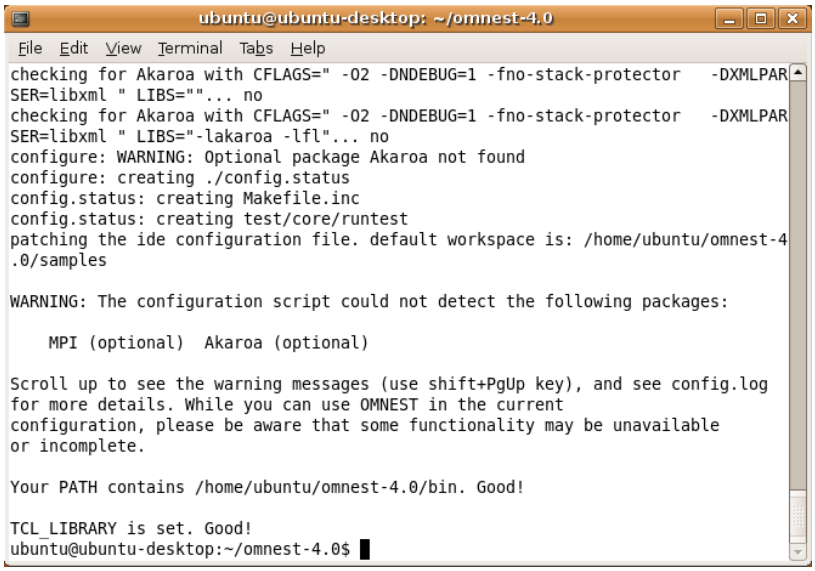

Figure 9.1. Configuring OMNEST

Normally, the `configure` script needs to be running under the graphical environment (X11) in order to test for `wish`, the Tcl/Tk shell. If you are logged in via an ssh session, or there is some other reason why X is not running, the easiest way to work around the problem is to tell OMNEST to build without Tcl/Tk. To do that, use the command

```
$ NO_TCL=1 ./configure
```

instead of plain `./configure`.

If there is an error during `configure`, the output may give hints about what went wrong. Scroll up to see the messages. (Use Shift+PgUp; you may need to increase the scrollback buffer size of the terminal and re-run `./configure`.) The script also writes a very detailed log of its operation into `config.log` to help track down errors. Since `config.log` is very long, it is recommended that you open it in an editor and search for phrases like *error* or the name of the package associated with the problem.

The `configure` script tries to build and run small test programs that are using specific libraries or features of the system. You can check the `config.log` file to see which test program has failed and why. In most cases the problem is that the script cannot figure out the location of a specific library. Specifying the include file or library location in the `configure.user` file and then re-running the `configure` script usually solves the problem.

When `./configure` has finished, you can compile OMNEST. Type in the terminal:

```
$ make
```

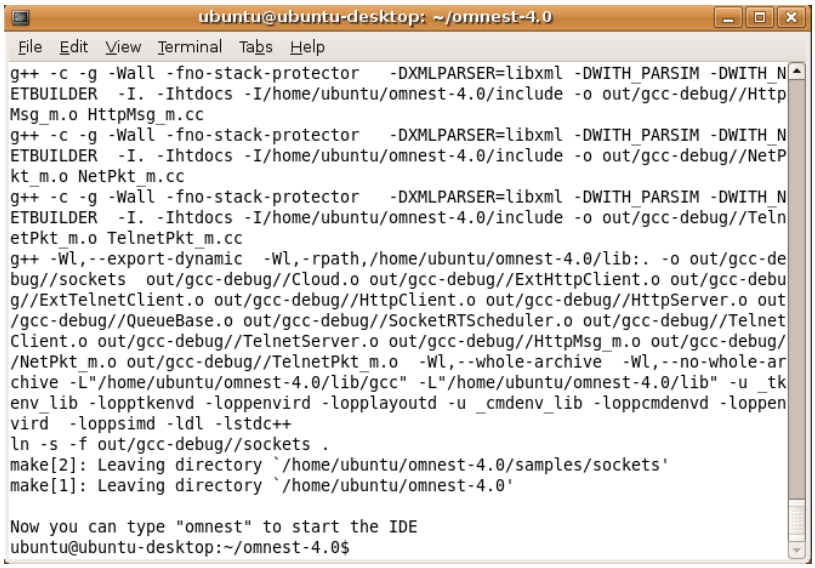

Figure 9.2. Building OMNEST

To take advantage of multiple processor cores, add the `-j2` option to the `make` command line.

The build process will not write anything outside its directory, so no special privileges are needed.

The make command will seemingly compile everything twice. This is because both debug and optimized versions of the libraries are built. If you only want to build one set of the libraries, specify `MODE=debug` or `MODE=release`:

```
$ make MODE=release
```

9.7. Verifying the Installation

You can now verify that the sample simulations run correctly. For example, the dyna simulation is started by entering the following commands:

```
$ cd samples/dyna
$ ./dyna
```

By default, the samples will run using the Tcl/Tk environment. You should see nice gui windows and dialogs.

9.8. Starting the IDE

The IDE is supported only on Windows, Mac OS X (x86) and Linux (x86,x64).

You can run the IDE by typing the following command in the terminal:

```
$ omnest
```

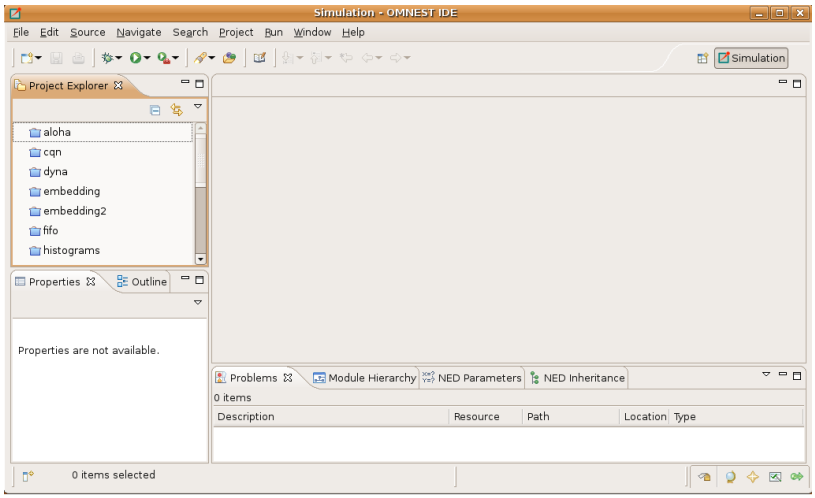

Figure 9.3. The Simulation IDE

If you would like to be able to access the IDE from the application launcher or via a desktop shortcut, run one or both of the commands below:

```
$ make install-menu-item
$ make install-desktop-icon
```

The above commands assume that your system has the `xdg` commands, which most modern distributions do.

9.9. Optional Packages

9.9.1. Akaroa

If you wish to use Akaroa, it must be downloaded, compiled, and installed manually before installing OMNEST.

As of version 2.7.9, Akaroa only supports Linux and Solaris.

Download Akaroa 2.7.9 from: http://www.cosc.canterbury.ac.nz/research/RG/net_sim/simulation_group/akaroa/download.chtml

Extract it into a temporary directory:

```
$ tar xfz akaroa-2.7.9.tar.gz
```

Configure, build and install the Akaroa library. By default, it will be installed into the `/usr/local/akaroa` directory.

```
$ ./configure
$ make
$ sudo make install
```

Go to the OMNEST directory, and (re-)run the `configure` script. Akaroa will be automatically detected if you installed it to the default location.

9.9.2. PCAP

The optional Pcap library allows simulation models to capture and transmit network packets bypassing the operating system's protocol stack. It is not used directly by

OMNEST, but OMNEST detects the necessary compiler and linker options for models in case they need it.

9.9.3. SystemC

To enable SystemC integration, add SYSTEMC=yes to the configure.user file, run *configure* and then rebuild your project. You can check the systemc examples in the samples/systemc-embedding directory.

Chapter 10. Build Options

10.1. Configure.user Options

The `configure.user` file contains several options that can be used to fine-tune the simulation libraries.

You always need to re-run the `configure` script in the installation root after changing the `configure.user` file.

```
$ ./configure
```

After this step, you have to remove all previous libraries and recompile OMNEST:

```
$ make cleanall
$ make
```

Options:

`<COMPONENTNAME>_CFLAGS`, `<COMPONENTNAME>_LIBS`
: The `configure.user` file contains variables for defining the compile and link options needed by various external libraries. By default, the `configure` command detects these automatically, but you may override the auto detection by specifying the values by hand. (e.g. `<COMP>_CFLAGS=-I/path/to/comp/includedir` and `<COMP>_LIBS=-L/path/to/comp/libdir -lnameoflib`.)

`WITH_PARSIM=no`
: Use this variable to explicitly disable parallel simulation support in OMNEST.

`WITH_NETBUILDER=no`
: This option allows you to leave out the NED language parser and the network builder. (This is needed only if you are building your network with C++ API calls and you do not use the built-in NED language parser at all.)

`NO_TCL=yes`
: This will prevent the build system to link with Tcl/Tk libraries. Use this option if your platform does not have a suitable Tcl/Tk package and you will run the simulation only in command line mode. (i.e. You want to run OMNEST in a remote terminal session.)

`EMBED_TCL_CODE=no`
: Tcl/Tk is a script language and the source of the graphical runtime environment is stored as `.tcl` files in the `src/tkenv` directory. By default, these files are not used directly, but are embedded as string literals in the executable file. Setting `EMBED_TCL_CODE=yes` allows you to move the OMNEST installation without caring about the location of the `.tcl` files. If you want to make changes to the Tcl code, you better switch off the embedding with the `EMBED_TCL_CODE=no` option. This way you can make changes to the `.tcl` files and

see the changes immediately without recompiling the OMNEST libraries.

`CFLAGS_[RELEASE/DEBUG]` — To change the compiler command line options the build process is using, you should specify them in the `CFLAGS_RELEASE` and `CFLAGS_DEBUG` variables. By default, the flags required for debugging or optimization are detected automatically by the `configure` script. If you set them manually, you should specify all options you need. It is recommended to check what options are detected automatically (check the `Makefile.inc` after running `configure` and look for the `CFLAGS_[RELEASE/DEBUG]` variables.) and add/modify those options manually in the `configure.user` file.

`LDFLAGS` — Linker command line options can be explicitly set using this variable. It is recommended to check what options are detected automatically (check the `Makefile.inc` after running `configure` and look for the `LDFLAGS` variable.) and add/modify those options manually in the `configure.user` file.

`SHARED_LIBS` — This variable controls whether the OMNEST build process will create static or dynamic libraries. By default, the OMNEST runtime is built as shared libraries. If you want to build a single executable from your simulation, specify `SHARED_LIBS=no` in `configure.user` to create static OMNEST libraries and then reconfigure (`./configure`) and recompile OMNEST (`make cleanall; make`). Once the OMNEST static libraries are correctly built, your own project have to be rebuilt, too. You will get a single, statically linked executable, which requires only the NED and INI files to run.

It is important to completely delete the OMNEST libraries (`make cleanall`) and then rebuild them, otherwise it cannot be guaranteed that the created simulations are linked against the correct libraries.

The following symbols can be defined for the compiler if you need backward compatibility with some older OMNEST 3.x features. They should be specified on the compiler command line using the `-DSYMBOLNAME` syntax. You can add these options to the `CFLAGS_RELEASE` or `CFLAGS_DEBUG` variables (e.g. `CFLAGS_RELEASE='-O2 -DNDEBUG=1 -DSYMBOLNAME'`).

`USE_DOUBLE_SIMTIME` — OMNEST 3.x used `double` as the type for simulation time. In OMNEST 4.0 and later, the simulation time is a fixed-point number based on a 64-bit integer.. If you want to work with `double` simulation time for some reason (e.g. during porting an OMNEST 3.x model), define the `USE_DOUBLE_SIMTIME` symbol for the compiler.

`WITHOUT_CPACKET` — In OMNEST 3.x, methods and data related to the modeling of network packets were included in the `cMessage` class. For these parameters, OMNEST 4.x has a new class called `cPacket` (derived from cMessage). If you want get back

the old behavior (i.e. having a single `cMessage` class only), define the `WITHOUT_CPACKET` symbol.

10.2. Moving the Installation

When you build OMNEST on your machine, several directory names are compiled into the binaries. This makes it easier to set up OMNEST in the first place, but if you rename the installation directory or move it to another location in the file system, the built-in paths become invalid and the correct paths have to be supplied via environment variables.

The following environment variables are affected (in addition to `PATH`, which also needs to be adjusted):

`OMNETPP_IMAGE_PATH` — This variable contains the list of directories where Tkenv looks for icons. Set it to point to the `images/` subdirectory of your OMNEST installation.

`OMNETPP_TKENV_DIR` — This variable points to the directory that contains the Tcl script parts of Tkenv, which is by default the `src/tkenv/` subdirectory of your OMNEST installation. Normally you don't need to set this variable, because the Tkenv shared library contains all Tcl code compiled in as string literals. However, if you compile OMNEST with the `EMBED_TCL_CODE=no` setting and then you move the installation, then you need to set `OMNETPP_TKENV_DIR`, otherwise Tkenv won't start.

`LD_LIBRARY_PATH` — This variable contains the list of additional directories where shared libraries are looked for. Initially, `LD_LIBRARY_PATH` is not needed because shared libraries are located via the *rpath* mechanism. When you move the installation, you need to add the `lib/` subdirectory of your OMNEST installation to `LD_LIBRARY_PATH`.

On Mac OS X, `DYLD_LIBRARY_PATH` is used instead of `LD_LIBRARY_PATH`. On Windows, the `PATH` variable must contain the directory where shared libraries (DLLs) are present.

10.3. Using Different Compilers

By default, the configure script detects the following compilers automatically in the path:

- Intel compiler (icc, icpc)
- GNU C/C++ (gcc, g++)
- Sun Studio (cc, cxx)
- IBM compiler (xlc, xlC)

If you want to use compilers other than the above ones, you should specify the compiler name in the `CC` and `CXX` variables, and re-run the configuration script.

Different compilers may have different command line options. If you use a compiler other than the default `gcc`, you may have to revise the `CFLAGS_[RELEASE/DEBUG]` and `LDFLAGS` variables.

10.4. Using Microsoft Visual C++

If you are using the Microsoft Visual C++ compiler on Windows, the build procedure is slightly different than the one used on Unix platforms.

The major differences are:

- The file `configuser.vc` must be used to specify your build options instead of `configure.user`
- There is no automatic configuration (`./configure`) for Visual C++ builds. You must review and specify each options in `configuser.vc` before compiling OMNEST.
- The build process is initiated with the command `nmake -f Makefile.vc` instead of `make`.