Regain Control of Growing Dependencies in OMNeT++ Simulations

Raphael Riebl · 3. September 2015
What can you expect?

- Status quo of OMNeT++ build process
- Why it can become insufficient
- Some basics about CMake
- How to combine OMNeT++ and CMake
Current build process for OMNeT++ projects

Invocation
- From IDE, see configuration stored in `.oppbuildspec`
- From custom Makefile or build script
opp_makemake usage from IDE
Inconvenient for complex setups

Example: Artery dependencies

🔗 Fork at github: https://github.com/riebl/artery
Inconvenient for complex setups

Example: Artery dependencies

Fork at github: https://github.com/riebl/artery
Why CMake?

- CMake is widely used for C/C++ projects
- Convenient user-interface for configuring builds (ccmake, cmake-gui)
- Working dependency handling
  - internal: correct build order within project
  - external: library and include directory locations (find_package)
- More accessible syntax compared to Makefiles
- Previous experience available 😊
CMake fundamentals

A CMake project consists of at least one `CMakeLists.txt`

- Defines executables and libraries to build
- Defines dependencies between these build targets and other libraries
CMake fundamentals

A CMake project consists of at least one *CMakeLists.txt*

- Defines executables and libraries to build
- Defines dependencies between these build targets and other libraries

Building a CMake project spans three phases

1. **Configuring** a build directory (with local CMake cache)
   Set custom compiler flags, determine location of external dependencies, select build type...

2. **Generating** files for a native build tool based on previous configuration
   GNU Makefiles, Ninja, Eclipse or Visual Studio projects etc.

3. **Building** with actual native build tool
Steps towards OMNeT++/CMake

1. Enhance *find_package* for OMNeT++
   - Directory with include headers
   - Import OMNeT++ libraries (debug and release)
   - Extract compiler flags from *Makefile.inc*
   - OMNeT++ message compiler
Steps towards OMNeT++/CMake

1. Enhance `find_package` for OMNeT++
   - Directory with include headers
   - Import OMNeT++ libraries (debug and release)
   - Extract compiler flags from `Makefile.inc`
   - OMNeT++ message compiler

2. Enable integration of existing OMNeT++ projects
   - Avoid making changes in foreign projects
   - Generic solution applicable to various projects is preferable
   - Should be easy to use
Steps towards OMNeT++/CMake

1. **Enhance `find_package` for OMNeT++**
   - Directory with include headers
   - Import OMNeT++ libraries (debug and release)
   - Extract compiler flags from `Makefile.inc`
   - OMNeT++ message compiler

2. **Enable integration of existing OMNeT++ projects**
   - Avoid making changes in foreign projects
   - Generic solution applicable to various projects is preferable
   - Should be easy to use

3. **Support OMNeT++ specific features, i.e. NED folders**
   - Additional CMake target property `NED_FOLDERS`
   - Targets inherit all NED folders of their dependencies automatically
   - Property value can be used for `opp_run` invocation
find_package(OmnetPP)

Implemented through *FindOmnetPP.cmake* located in CMake's module path. Basically, OMNeT++ can be treated like any other C/C++ library.

- Additional location hint by looking up *omnetpp* binary in PATH.
- Extract information from *Makefile.inc* with regular expressions.
- Dedicated libraries with debug symbols are available.
Integrating OMNeT++ legacy projects

Exploit Makefile generated by opp_makemake, e.g. inet/src/Makefile (INET 3.0)

# OMNeT++/OMNEST Makefile for libINET
#
# This file was generated with the command:
# opp_makemake -f --deep --make-so -o INET -O out -pINET
  --no-deep-includes -Xinet/applications/voipstream
  -Xinet/linklayer/ext -Xinet/transportlayer/tcp_lwip
  -Xinet/transportlayer/tcp_nsc -I../src -DWITH_TCP_COMMON
  -DWITH_TCP_INET -DWITH_IPv4 -DWITH_IPv6 -DWITH_xMIPv6
  -DWITH_GENERIC -DWITH_FLOOD -DWITH_UDP -DWITH_RIP -DWITH_SCTP
  -DWITH_DHCP -DWITHETHERNET -DWITH_PPP -DWITH_MPLS -DWITH_OSPFv2
  -DWITH_BGPv4 -DWITH_PIM -DWITH_RIP -DWITH_POWER -DWITH_RADIO
  -DWITH_AODV -DWITH_MANET -DWITH_IEEE80211 -DWITH_APSKRADIO
  -DWITH_IDEALWIRELESS -DWITH_TUN -DWITH_BMAC -DWITH_LMAC
  -DWITH_IEEE802154 -DWITH_CSMA
#
#
1 Parse opp_makemake line with a helper script opp_cmake
2 Create a CMake file with IMPORTED CMake targets of legacy project
Integrating OMNeT++ legacy projects

Overview of involved tools during build phases

Import legacy OMNeT++ dependency in CMake project

<table>
<thead>
<tr>
<th>Legacy</th>
<th>opp_makemake</th>
<th>Makefile</th>
<th>legacy binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMake generate</td>
<td>opp_cmake</td>
<td>legacy-targets.cmake</td>
<td>project binary</td>
</tr>
<tr>
<td>CMake build</td>
<td>import_opp_target</td>
<td>compiling &amp; linking</td>
<td>project binary</td>
</tr>
</tbody>
</table>
An OMNeT++ example project using CMake

```cpp
project(YourProject)
cmake_minimum_required(VERSION 3.0)
set(CMAKE_MODULE_PATH ${PROJECT_SOURCE_DIR}/cmake)

find_package(OmnetPP 4.6 REQUIRED)
# definition of add_opp_run and import_opp_target macros omitted

find_path(INET_DIR NAMES src/inet/package.ned DOC "INET root directory")
import_opp_target(inet ${INET_DIR}/src/Makefile)

set(SOURCES src/a.cc src/b.cc)
add_library(project_library SHARED ${SOURCES})
set_property(TARGET project_library PROPERTY NED_FOLDERS src)
target_link_libraries(project_library opp_interface inet)

add_opp_run(run_project omnetpp.ini project_library)
```
What's next?

Presented CMake macros, scripts and example project are available at https://github.com/riebl/artery/releases/tag/opp-summit2015

⚠️ Proof-of-Concept is working for Artery
✉️ Might it be valuable (reusable) for your simulation?
🗣 Feedback is welcome as are suggestions for improvement!