Towards a modern CMake workflow

OMNeT++ Community Summit

Heinz-Peter Liechtenecker, Raphael Riebl
8 – 10 September 2021
Introduction

Motivation for another build system

CMake as a powerful alternative

Developing with Visual Studio Code

Technical Preview
Motivation for another build system
The OMNeT++ build system

OMNeT++ ...

- comes with an Eclipse-based IDE
- comes with a pre-built MinGW/MSYS environment (Linux build tools for Windows)
- offers an “out of the box” experience (IDE, toolchain, examples) for beginners

OMNeT++ Makefile-based build system (opp_makemake) ...

- feels native only on Unix systems
- makes management of dependencies and variants difficult
- often conflicts with CMake-based dependencies
- complicates the use of other IDEs (especially on Windows)
The OMNeT++ build system

OMNeT++ ... 

- comes with an Eclipse-based IDE
- comes with a pre-built MinGW/MSYS environment (Linux build tools for Windows)
- offers an “out of the box” experience (IDE, toolchain, examples) for beginners

OMNeT++ Makefile-based build system (opp_makemake) ... 

- feels native only on Unix systems
- makes management of dependencies and variants difficult
- often conflicts with CMake-based dependencies
- complicates the use of other IDEs (especially on Windows)
The OMNeT++ build system

OMNeT++ ... 

- comes with an Eclipse-based IDE
- comes with a pre-built MinGW/MSYS environment (Linux build tools for Windows)
- offers an “out of the box” experience (IDE, toolchain, examples) for beginners

OMNeT++ Makefile-based build system (opp_makemake) ... 

- feels native only on Unix systems
- makes management of dependencies and variants difficult
- often conflicts with CMake-based dependencies
- complicates the use of other IDEs (especially on Windows)
The OMNeT++ build system

OMNeT++ ... 

- comes with an Eclipse-based IDE
- comes with a pre-built MinGW/MSYS environment (Linux build tools for Windows)
- offers an "out of the box" experience (IDE, toolchain, examples) for beginners

OMNeT++ Makefile-based build system (opp_makemake) ... 

- feels native only on Unix systems
- makes management of dependencies and variants difficult
- often conflicts with CMake-based dependencies
- complicates the use of other IDEs (especially on Windows)
The OMNeT++ build system

OMNeT++ ...

- comes with an Eclipse-based IDE
- comes with a pre-built MinGW/MSYS environment (Linux build tools for Windows)
- offers an “out of the box” experience (IDE, toolchain, examples) for beginners

OMNeT++ Makefile-based build system (opp_makemake) ...

- feels native only on Unix systems
- makes management of dependencies and variants difficult
- often conflicts with CMake-based dependencies
- complicates the use of other IDEs (especially on Windows)
OMNeT++ build system

- comes with an Eclipse-based IDE
- comes with a pre-built MinGW/MSYS environment (Linux build tools for Windows)
- offers an “out of the box” experience (IDE, toolchain, examples) for beginners

OMNeT++ Makefile-based build system (opp_makemake)

- feels native only on Unix systems
- makes management of dependencies and variants difficult
- often conflicts with CMake-based dependencies
- complicates the use of other IDEs (especially on Windows)
The OMNeT++ build system

OMNeT++ ... 

- comes with an Eclipse-based IDE
- comes with a pre-built MinGW/MSYS environment (Linux build tools for Windows)
- offers an “out of the box” experience (IDE, toolchain, examples) for beginners

OMNeT++ Makefile-based build system (opp_makemake) ...

- feels native only on Unix systems
- makes management of dependencies and variants difficult
- often conflicts with CMake-based dependencies
- complicates the use of other IDEs (especially on Windows)
We believe
A well-designed software architecture and build environment
reduces the management overhead and thus allows even small
teams to maintain and improve complex projects.

1. CMake is the de facto standard for almost every C/C++
   open-source project thanks to its versatility

2. CMake generates native build environments that will compile
   source code, create libraries, generate wrappers and build
   executables in arbitrary combinations

3. CMake is cross-platform from the beginning (e.g., can
   generate Makefiles but also the Ninja build rules)
Why CMake?

We believe
A well-designed software architecture and build environment reduces the management overhead and thus allows even small teams to maintain and improve complex projects.

1. CMake is the de facto standard for almost every C/C++ open-source project thanks to its versatility

2. CMake generates native build environments that will compile source code, create libraries, generate wrappers and build executables in arbitrary combinations

3. CMake is cross-platform from the beginning (e.g., can generate Makefiles but also the Ninja build rules)
We believe
A well-designed software architecture and build environment reduces the management overhead and thus allows even small teams to maintain and improve complex projects.

1. CMake is the de facto standard for almost every C/C++ open-source project thanks to its versatility
2. CMake generates native build environments that will compile source code, create libraries, generate wrappers and build executables in arbitrary combinations
3. CMake is cross-platform from the beginning (e.g., can generate Makefiles but also the Ninja build rules)
Why CMake?

We believe
A well-designed software architecture and build environment reduces the management overhead and thus allows even small teams to maintain and improve complex projects.

1. CMake is the de facto standard for almost every C/C++ open-source project thanks to its versatility

2. CMake generates native build environments that will compile source code, create libraries, generate wrappers and build executables in arbitrary combinations

3. CMake is cross-platform from the beginning (e.g., can generate Makefiles but also the Ninja build rules)
CMake as a powerful alternative
From Sources to Executables

ONE DOES NOT SIMPLY
GET AN EXECUTABLE FROM SOURCE CODE
Requirements

- **Write code only once** – if the concept performs in simulation, have everything ready for the production code
- Minimize management overhead by having a single source of truth/configuration (CMake files)
- Allow for continuous integration (automated tests etc.)
- Seamless workflow between simulation and actual production code (transferring results made easy)
Requirements

- Write code only once – if the concept performs in simulation, have everything ready for the production code
- **Minimize management overhead** by having a single source of truth/configuration (CMake files)
- Allow for continuous integration (automated tests etc.)
- Seamless workflow between simulation and actual production code (transferring results made easy)
Requirements

- Write code only once – if the concept performs in simulation, have everything ready for the production code.
- Minimize management overhead by having a single source of truth/configuration (CMake files).
- Allow for continuous integration (automated tests etc.).
- Seamless workflow between simulation and actual production code (transferring results made easy).
Requirements

- Write code only once – if the concept performs in simulation, have everything ready for the production code
- Minimize management overhead by having a single source of truth/configuration (CMake files)
- Allow for continuous integration (automated tests etc.)
- **Seamless workflow** between simulation and actual production code (transferring results made easy)
OMNeT++ is neatly integrated into sophisticated projects:

- **Simulation**: business logic and INET\(^1\)
- **Testing**: Unit tests covering your code, e.g., with GTest
- **Production**: business logic and ASIO\(^2\) for deployment

\(^1\)OMNeT++ model, i.e., simulated network communications
\(^2\)C/C++ asynchronous network library, i.e., native network communications
Developing with Visual Studio Code
• Straightforward workflow for novices and self-contained simulation models, i.e. without dependencies to third-party components
• Does not ship with CMake support (Eclipse plugin exists)
• Its Eclipse core can be slow and bulky
What does the community want?³

³Stackoverflow Survey 2021:
https://insights.stackoverflow.com/survey/2021
VSCode is cross-platform and highly customisable. We suggest the following extensions making VSCode a neat IDE for OMNeT++ development (with CMake):

- Cpptools (C/C++ Language Support)
- CMake (Language Support)
- CMake Tools (CMake project integration and automation)
- OMNeT++-NED (NED language support)
- VSCode-LLDB (LLDB debugging support)
Technical Preview
OMNeT++ tictoc demo in VSCode
https://github.com/HpLightcorner/opp-summit-2021-cmake-tp
Find the latest OMNeT++ CMake package at
https://github.com/omnetpp/cmake