OMNeT++ Goes Python

András Varga
andras@omnetpp.org
OMNeT++ Core Team
Introduction

• After finishing OMNeT++ 6.0...

• We started playing around with Python because we saw huge potentials there
  – enthused by the Python-powered Analysis Tool
  – during INET development and maintenance, a Python REPL with a home-grown lib has grown to be an indispensable tool
  – discovered the cppyy C++ FFI Python package
(Snapshot from the office)
Areas where Python can be useful

- Setting up models (e.g. topology building)
- Extending the capabilities of NED expressions
- Implementing simulation-specific ad-hoc components (e.g. scenario managers, custom traffic generators, etc.)
- Simulation control (e.g. custom stop conditions)
- Managing and running simulations and simulation campaigns (workflow automation)
- Result analysis (inside and outside of the IDE)

Disadvantages of Python in OMNeT++
- Speed (or lack thereof)
  - you want to keep Python code out of the “hot” parts of the model
- The word “module” has become ambiguous
  - do you mean “importable Python file”, or “OMNeT++ simulation component”? ;-)

OMNeT++ Community Virtual Summit – November 2-3, 2022
Upcoming Python-related tools

- **Python in NED expressions**
  - `pyeval()` and `pycode()` NED functions
- **Modules implemented in Python**
  - `@pythonClass` NED property
  - modules can be written from scratch, or subclassed from existing (e.g. INET) modules
- **Simulation library as a Python package**
  - `from omnetpp.runtime import *`
  - based on cppyy
- **Python package for processing simulation results**
  - `from omnetpp.scave import results, ...`
- **Python package for managing and running simulations**
  - `from omnetpp.simulation import *`
The `pyeval()` and `pycode()` NED functions

- **`pyeval(<expr>,...)`** - evaluates a Python expression string
  - `pyeval("2*3")`
  - `pyeval("x: 2*x", 3) --> 6`

- **`pycode(<block>,...)`** - evaluates a Python statement block
  - block must end in a "return" statement (like a function body)
  - `pycode("import math; return math.factorial(15)")`
  - `pycode("a,b: import math
  if a<0 or b<0: return math.nan
  return math.gcd(a,b)", 70, 62)`

More details about `pyeval()`/`pycode()`, @pythonClass and cppyy in Attila Török’s Summit presentation: *Using Python within Simulations*
The @pythonClass NED property

- Denotes that module is implemented by a Python class

```plaintext
// sink.ned
simple Sink {
  parameters:
    @pythonClass;
    ...
}

# sink.py
from omnetpp.runtime import *

class Sink(omnetpp.cSimpleModule):
  def handleMessage(self, msg):
    ...
```
The cppyy package

• Dynamic runtime Python bindings for C++
  – allows cross-inheritance and callbacks, template instantiation and more
  – simple example:

  ```python
  import cppyy
  cppyy.include("iostream")
  cppyy.cppdef("""class A { public: void sayHello() { std::cout << "Hello" << std::endl; } };""")
  A = cppyy.gbl.A
  a = A()
  a.sayHello()  # prints “Hello”
  ```

• Foundations:
  – Cling, the interactive C++ interpreter from CERN
  – Cling itself builds on Clang and LLVM
  – (at least this is what they want you to believe, but actually it relies on magic)
The omnetpp.runtime package

- Based on **cppyy**, it exposes the simulation library as a Python package
  - both simulation kernel and the “envir” infrastructure
  - its essence:
    ```python
    import cppyy
cppyy.include("omnetpp.h")
    ```

- OMNeT++ is undergoing extensive refactoring:
  - Reusability of “envir” part improved
  - Python readiness
  - Multi-thread support
    - Allow multiple threads to be used for simulation (but simulations are still *single-threaded!*)
      ```
      ./aloha -u Cmdenv --cmdenv-num-threads=8 ...
      ```
    - Global variables became `thread_local` or simulation-scope (`cSimulation::getSharedVariable<T>(name) - new`)
The omnetpp.scape package

- Part of OMNeT++ 6.0

Analysis API

- Chart scripts usually begin with:
  - `from omnetpp.scape import results, chart, plot, utils, vectorops as ops`
- Terminology:
  - “chart” is what you edit (Python script + configuration)
  - “plot” is the artifact created by running the “chart”
- The packages:
  - `results`: Querying results into Pandas data frames
  - `chart`: Access to chart properties
  - `plot`: Plot to the IDE native plot widget
  - `utils`: Common interface to Matplotlib and native widgets; misc utility functions
  - `vectorops`: Vector operations (window average, running sum, etc)
  - `analysis`: Read/write/run ANF files from standalone scripts

Slide from last year’s Summit
The omnetpp.simulation package

- Library and toolset for managing and running simulations and campaigns in various ways
  - in-process / local / distributed (e.g. ssh cluster using Dask)
  - for results, for regression testing (e.g. fingerprints), etc.
- Grown from the needs of INET development and maintenance
  - “run all simulations, utilizing all computing resources I have access to”
  - “refresh list of simulations to be fingerprint-tested”
  - “re-run failing fingerprint tests”
  - “re-run simulations that contain WiFi” (after WiFi model change)
  - “compare results to those with INET version X”
- Human time is expensive:
  - automate/assist as much as possible (high-level tools, REPL, etc)
  - store instead of recompute (fingerprints, simulation results, etc)
  - utilize available computing resources (multi-core, ssh cluster, etc)
DEMO
Recap of the Demo

Python added to several OMNeT++ sample simulations:
• pyfifo: pure Python simple modules
• routing: various examples for making use of Python
• aloha: examples for running simulations from Python

To try, use the OMNeT++ version at the following URL:
https://drive.google.com/file/d/1jDD-vtzi9YVzShrw2fP6q1SUU5VMVsDM/view?usp=share_link
samples/pyfifo

- A fairly verbatim re-implementation of the `fifo` example
- With all modules written in Python
samples/routing

Configurations in omnetpp.ini:

- [FromCsv]: Setting up a network specified in CSV, using Python
- [RandomTree]: Setting up a network generated with NetworkX Python package
- [DumbbellFaultyLink]: Shows a scenario manager written in Python
- [AppExt]: Extending a C++ simple module from Python
- [App2]: Parameter values produced using pyeval()/pycode()
samples/aloha

Python scripts:

- **example1.py**: instantiating a simulation
  
  ```python
  simulation = omnetpp.cSimulation(...)
  ```

- **example2.py**: all simulations in an omnetpp.ini parameter study
  
  ```python
  ini.getNumRunsInConfig("PureAlohaExperiment")
  ```

- **example3.py**: manually organizing a parameter study
  
  ```python
  for numHosts in [10,15,20]:
      for iaMean in [1,2,3,4,5,7,9]:
          ...
  ```

- **example4.py**: utilizing multiple CPU cores
  
  ```python
  with multiprocessing.Pool() as p:
      p.map(alohaJob, taskList)
  ```

- **example5.py**: simulation-based optimization
  
  ```python
  scipy.optimize.minimize(...)  
  ```

- **example5a.py**: preserve and also plot all simulations
Questions?