OMNeT++ Virtual Community Summit 2022, Nov. 2-3.

Using Python within OMNeT++ simulation models

Attila Török

Agenda

- cppyy
  - provides fully automatic, dynamic Python-C++ bindings
- Implementing simple module logic in Python
  - `@pythonClass`

cppyy

- Background: generates Python bindings for C++ dynamically
- Based on LLVM, clang, cling
- Used by ROOT at CERN

https://github.com/wlav/cpycppyy/
https://cppyy.readthedocs.io/

cppyy usage

```
import cppyy

cppyy.cppdef("""
  void hello() {}
  class Base { };
""")

cppyy.gbl.hello()
b = cppyy.gbl.Base()
class Derived(cppyy.gbl.Base):
   pass
```

Ownership management

- Python: GC (reference counting)
- C++: explicit memory management
  - OMNeT++ ownership tree, soft owner…
- `object *f()`:
  - “Is it given or just shown to me?”
- `void f(object *)`:
  - “Am I giving it away or just showing it?”
- `__python_owns__` flag

cppyy pythonizations

- Helps make C++ code more idiomatic Python
  - Make container objects easily iterable
  - Customize string representation of objects
  - Make friend operators work
- Ownership management:
  - Wrap methods and set `__python_owns__` on parameters and return values as needed

cppyy pythonizations

```
def setsend_msg_ownership(klass, name):
    if name == "cSimpleModule":
        orig_send = klass.send
        def new_send(self, msg):
            msg.__python_owns__ = False
            return orig_send(self, msg)
        klass.send = new_send

cppyy.py.add_pythonization(setsend_msg_ownership, 'omnetpp')
```

`@pythonClass` details

- `@pythonClass` is module property
- Marks simple modules as implemented in Python
- Entirely in Python, or extending a C++ base class

@pythonClass;

- .py file next to the .ned file, with the same name
- Class name same as module type name

Sources.ned:

```
simple PeriodicSource {
    @pythonClass;
}
```

Sources.py:

```
class PeriodicSource(omnetpp.cSimpleModule):
    …
```

@pythonClass(Foo)

- .py file next to the .ned file, with the same name
- Class name is given

Sources.ned:

```
simple PeriodicSource {
    @pythonClass(PySource);
}
```

Source.py:

```
class PySource(omnetpp.cSimpleModule):
    …
```

@pythonClass(mod.Foo)

- Fully qualified Python class name
  - NED path entries also PYTHONPATH entries

Source.ned:

```
simple PeriodicSource {
    @pythonClass(simplemodules.PySource);
}
```

simplemodules.py:

```
class PySource(omnetpp.cSimpleModule):
    …
```

@pythonClass(mod.foo.Bar)

- Fully qualified Python class name
  - NED path entries also PYTHONPATH entries

Source.ned:

```
simple PeriodicSource {
    @pythonClass(modules.sources.PySource);
}
```

modules/sources.py:

```
class PySource(omnetpp.cSimpleModule):
    …
```

Why cppyy?

- Dynamic binding generation, less manual work
  - Still requires some fine tuning

https://cppyy.readthedocs.io/en/latest/philosophy.html

- Alternatives: pybind11 (+Binder), nanobind, Boost.Python, Shiboken

cppyy implications

- Model headers need to be available at runtime
- Startup time increases (by ~1s)
- Runtime cost every time, even for “built-ins”
- Can crash if not used carefully
  - e.g. GC deletes object prematurely (ownership…)
  - Prints stack trace as help

cppyy quirks

- Calling protected base methods in overrides
  - Needs an intermediate “publicist” class
- Exceptions in overloaded methods
  - Thinks that overload resolution failed

https://github.com/wlav/cppyy/issues/

Thank you!

Obligatory last empty slide