OMNet++
What already exist

Chang-Gun Lee (cglee@snu.ac.kr)

Assistant Professor

The School of Computer Science and Engineering

Seoul National University

Comprehensive Documents

- OMNet++ 3.2 Manual (not 3.3. Why?)
  - doc/usman.pdf
  - http:/www.omnetpp.org/doc/manual/usman.html
- OMNet++ 3.2 API Reference
  - http://www.omnetpp.org/doc/api/index.html

NED Language (0)

- define the followings
  - **import** other .ned files
  - **channel** definition
  - **simple** module definitions
  - compound **module** definitions
  - **network** definitions (usually just one network)

NED Language (1)

- “import” directive
  - import other .ned files
  - modular structure of system architecture
- channel definition
  - The channel name can be used later in the NED description to create connections with these parameters
  - can define various (optional) attributes: delay, error, data rate

```
channel ChannelName
        delay 0.01 // sec
        error 1e-8
        datarate 128000 // buts/sec
endchannel
```

NED Language (2)

- Simple module definition
  - parameters: e.g., interarrivalTime, numOfMessages, address, etc.
    - parameters can be assigned from NED (when the module is used as a building block of a larger compound module) or from the config file “omnetpp.ini” or from user input
  - gates: in or out types
    - also can define gate array (e.g., in: output[];)

```
simple SimpleModuleName
        parameters: //….
        gates: //…
endsimple
```

NED Language (3)

- Compound module definition
  - parameters
  - gates
  - submodules: instantiate submodules, assigning their parameters
  - connections: direct connections of submodule gates or connections through channels

```
module CompoundModuleName
        parameters: //….
        gates: //…
        submodules: //…
        connections: //…
endmodule
```

NED Language (4)

- Network definitions
  - **module** just defines a module type.
  - **network** creates an instance of the module type
  - only a compound module type “without gates” can be used in a network defintion

network *TopLevelCompoundModuleName: NetworkName*
    parameters: //….
endnetwork

Simple Module Implementation

1. void initialize
   - prepare simulation of the module
2. void handleMessage(cMessage *msg)
   - describe detail operations for occurring events
   - core that specifies the module's behavior
   - implement using Message/event related functions
     - send(), scheduleAt() (self-message), cancelEvent()
     - no need to use "receive()" and "wait" since OMNet++ kernel will call the module's handleMessage whenever an event occurs
3. void finish()
   - wrap-up simulation by summarizing measurements and producing reports

Dynamic module creation

- When we need that?
  - a mobile node enters and leaves the simulated area
  - TCP connections dynamically created and destroyed
- How to create?
  - get module type first
    - cModuleType *modType = findModuleType(“WirelessNode”)
  - instantiate the module
    - mod = modtype->createScheduleInit(“node”, this);
    - It does creat() + buildInside()+scheduleStart(now)+callInitialize()
- How to destroy?
  - mod->deleteModule()

Connecting dynamic modules

- How to create connections?
  - cModuleType *modType = findModuleType(“WirelessNode”);
  - cModule *a = modType->createScheduleInit(“a”, this);
  - cModule *b = modType->createScheduleInit(“b”, this);
  - a->gate(“out”)->connectTo(b->gate(“in”)); //srcGate->connectTo(destGate)
  - b->gate(“out”)->connectTo(a->gate(“in”));
- How to remove connections?
  - a->gate(“out”)->disconnect(); // srcGate->disconnect()
  - b->gate(“out”)->disconnect();

Creating My Own Message

- Basically, we have to subclass cMessage class
- More convenient way is to
  - make myMessage.msg file including

```
message myMessage
{
        fields:
                int srcAddress;
                int destAddress;
                int hops = 32;
}
```

  - let the message subclass compiler generate C++ classes (myMessage_m.h, myMessage_m.cc)

see API References

- **Simulation core classes**
- **Container classes**
- **Random number generation**
- **Statistical data collection**
- **Utility classes**
- **User interface: cEnvir and ev**
- **Enums, types, function typedefs**
- **Functions**
- **Macros**
- **Internal classes**
- **Extension interface to Envir**
- **Parallel simulation extension**

Container Classes

- Queue class: cQueue
  - double-linked list to store elements of cObject (almost all classes of OMNet+ library)
  - useful member functions
    - queue.insert(msg) (also, insertBefore(), insertAfter())
    - msg = queue.pop(); (to remove a specific item, queue.remove(msg))
    - queue.empty();
    - queue.length();
  - Also, possible to implement a priority queue
- Expandable array: cArray
  - Automatically grows when full
  - useful member functions
    - array.add(p); (also addAt(5,p)
    - int index = array.find(p)
    - array.remove(p);

Refer to manual and API references whenever needed!