Tython
Simulation Scripting for TinyOS

Mike Demmer / Phil Levis
NEST Retreat
January 2004
Why is simulation important?

• Motivating example: A first year graduate student in a sensor networks class implementing a simple TDMA slotted ring protocol (guess who)

• What did I want from simulation?
  - more manageable code/build/test/debug cycle
  - richer debugging output - not limited to a few LEDs
  - ramp up complexity, e.g. start with a perfect radio

• And more specifically:
  - move motes in and out of range of each other
  - fail certain motes to make sure protocol can handle it
  - test one way radio connectivity
  - ensure basic correctness while adding more complexity
What tools did I have to use?

- **TOSSIM**
  - discrete event simulator for TinyOS applications
  - same program source that runs on the mote hardware but instead compiled into a simulator executable
  - bit-level radio model, simulated ADC values

- **TinyViz**
  - framework to visualize and manipulate TOSSIM executions via a Java based application GUI
  - adds dynamics - can turn motes on/off, affect the radio and sensor models, move motes around, etc.
  - Java Plugin API to add custom visualizations / manipulations
What's wrong with these tools?

- **Interactivity only through GUI**
  - cumbersome to do a repeated test such as moving motes into and out of range
  - can't reproduce operations from run to run, thus hard to isolate differences

- **Extensibility only through Java Plugins**
  - relatively low level API
  - better suited to extending the TinyViz tool than as a mechanism to enable experimentation

- **No mechanism to access mote state**
  - main source of program output is by adding debug messages to the application source
So what’s the solution?

• Add a scripting framework
  - integrate *Jython* - a Java implementation of the Python programming language with object reflection

• Restructure TinyViz internals into *SimDriver*
  - core components to manage interaction such as the radio model, mote location, communication with TOSSIM, etc.
  - optional GUI for visualization

• Provide manipulation primitives
  - Jython object reflection used to expose the SimDriver core
  - extensible library of Python routines and objects for more complex functionality

• Add accessibility to mote frame variables
  - NesC compiler now generates a variable resolution function, accessed via the TOSSIM command interface
How do TinyViz / TOSSIM relate?

TinyViz

- TinyViz GUI
- Plugins
- SimComm

Event Bus

TOSSIM

- Mote Application Program
- Event Queue
- Radio Model

External Comm

commands

events
How does scripting fit in?

SimDriver

- TinyViz GUI
- Plugins
- ScriptInterpreter
- simcore
- Reflected Classes
- simutil
- Python Classes
- User Script
- Event Bus

SimComm

- commands
- events

External Comm

- Mote Application Program
- Radio Model
- Event Queue

NesC

TOSSIM

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script

SimDriver

TinyViz GUI

Plugins

ScriptInterpreter

simcore

Reflected Classes

simutil

Python Classes

User Script

SimComm

Event Bus

commands

events

NesC

TOSSIM

Mote Application Program

Radio Model

Event Queue

User

Script
How about an example...

```python
###
### Periodic Beacon Test Script
###
import simcore, simutil, simtime
import net.tinyos.message

# boot the 10 motes
for i in range(0, 9):
    simcore.motes[i].turnOn()

# function to send a new route update message to mote 0
def route_update(event):
    msg = net.tinyos.message.MyRouteUpdateMsg()
    simcore.comm.sendRadioMessage(0, simcore.getTosTime(), msg)

# set up a repeated call to inject the packet every 10 seconds
simutil.Periodic(simtime.onesec * 10, route_update)

# run for five minutes, then quit the simulation
simutil.CallIn(simtime.onemin * 5, simcore.sim.exit)
```
What's going on behind the scenes?

- **Jython reflects Java classes to Python**
  - hence the MIG generated `MyRouteMessage` class

- **The `simcore` module provides a Python object interface to hide SimDriver / TOSSIM complexities**
  - e.g. `sendRadioMessage` hides details of creating a `TosMSG` wrapper and sending a `RadioMsgSendCommand` to TOSSIM
  - building blocks for more complex functionality (e.g. `simutil`)

- **TOSSIM event queue is the main control loop**
  - all TOSSIM events are sent to the SimDriver and internally distributed via the Event Bus
  - periodic / future events implemented by inserting an event and registering a callback handler
  - python scripts can register a handler to get events as well
What about mote state variables?

• When debugging, it’s useful to probe around program state, as is done with gdb

• Scripts can also use this information
  - e.g. move a mote randomly until it comes in radio range of another mote, then stop

• NesC generates a lookup table to resolve component frame variables
  - translates logical variable name into memory address / size

• TOSSIM exports resolve/fetch/set commands
  - reflected through the simcore object interface
How does that fit in?

SimDriver
- TinyViz GUI
- Plugins
- ScriptInterpreter
- MoteVariables
- simcore
  - Reflected Classes
- simutil
  - Python Classes
- User Script

Event Bus

SimComm

TOSSIM
- Mote Application Program
- Radio Model
- Variable Resolver

External Comm

Event Queue

commands

events
One more example...

// NesC application source:
module NeighborCountM {}  
implementation {
    int count;
    command result_t StdControl.init() {}
}

# Python script:
import simcore, simutil, simtime

def displaycount():
    mote = simcore.motes[0]
    count = mote.getInt("NeighborCountM$count")
    mote.setLabel("%d neighbors" % count)

    simutil.Periodic(simtime.onesec * 5, displaycount)
So what does all this let me do?

- **Repeatable simulation dynamics**
  - move motes around, inject radio beacons, etc...
  - framework to do comparison experiments

- **Automatic parameter exploration**
  - set up a set of scripted tests as a batch job

- **“Adversarial” simulations**
  - force edge conditions such as one way connectivity, failing motes, bogus ADC values, etc.

- **Interactive scripting console for debugging**
  - pause / restart simulation, probe state
  - can probe a running simulation, attach / detach, etc
To wrap up

• Tython has been checked into the main TinyOS repository, and will be part of the 1.1.4 release

• More information and a simple demo at the poster session later on tonight

• Questions?
Backup Slides
Why is simulation important?

• Sensor Networks programming is hard
  - distributed, event driven systems, limited output
  - prototyping, protocol experimentation cumbersome

• Comparisons of related protocols and algorithms
  - isolate variability of the environment for “fair” trials

• Automatic exploration of parameter space
  - batch simulation execution to tune applications

• Regression testing
How does this callback stuff work?

The *Periodic* and *CallIn* classes are implemented by:

a) registering an event handler with the internal SimEventBus, allocating a unique event identifier

b) queuing an event callback in the future by inserting an event on the TOSSIM event queue with the given id

c) executing the callback closure if the event's id matches
import simcore
import net.tinyos.sim.event

class Periodic:
    def __init__(self, interval, callback):
        # get a unique event identifier
        pauseID = simcore.comm.getPauseID()

        # define an event handler to execute the callback if the id matches
        def mycallback(event):
            if (event.get_id() == pauseID):
                callback(event)
                simcore.sim.pauseInFuture(event.getTime() + interval, pauseID)

        # register a handler to get SimulationPausedEvent events
        evclass = net.tinyos.sim.event.SimulationPausedEvent
        simcore.interp.addEventHandler(mycallback, evclass)

        # enqueue the callback after interval seconds
        now = simcore.sim.getTossimTime()
        simcore.sim.pauseInFuture(now + interval, pauseID)