MuDiSP3 and NePSing simulation environments: link and network level C++ simulation engine for satellite systems

Luca Simone Ronga
Tommaso Pecorella
CNIT
The INeSiS project

- DSP (synchronous data flow) and Network Protocol (Discrete Event) simulations
- Full support for user defined simulation parameters
- Availability of statistical analysers
- Self contained simulation executable that allows background executions and multi processing
- High efficiency and execution speed
- Textual user interface (telnet remote access)

Integrated Network Protocols and Signal Processing Simulator

<table>
<thead>
<tr>
<th>MuDiSP3</th>
<th>NePSing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multirate Digital Signal Processor</td>
<td>Network Protocols Simulator</td>
</tr>
</tbody>
</table>
INeSiS architecture

INeSiS

MuDiSP3
NePSing

Statistical analyser
Random generators
Parameter manager

Library Blocks

User Code

External Libraries
TNT, MTL, ...

TD02-034-S
class MyBlock : public Block {

public:

    InPort <double> in1;     // input port definition
    OutPort <double> out1;   // output port definition

    MyBlock();

    void Setup();  // executed once initially
    void Run();    // every iteration
    void Finish(); // executed at the end

};
MuDiSP3

- Events happen synchronously (e.g. DSP)
- Multi-rate w automatic execution sequence discovery
- Block based system representation
  - primitive blocks whose behavior is defined by a segment of C++ code,
  - compound blocks composed by other interconnected blocks inside.
- Block has 4 basic behaviors:
  - **Build()** contains the sub-blocks connection instructions, it is executed once per simulation.
  - **Setup()** contains the code for initializing the processing of the block and permits the reading of the run-time parameters; it is also executed once per simulation.
  - **Run()** contains the code of the actual processing of the block, it usually contains instructions for collecting data from the input ports of the block and/or instructions for delivering processed data to the output ports. It is executed once per iteration.
  - **Finish()** it performs all the post-simulation processing, organizing, displaying or saving results. It is executed once.
- Parameter files avoid re-compiling the simulation target
NePSing – Discrete Events

Devices (C++)

Events (C++)
NePSing

• A discrete event (DE) simulation is composed of a set of *device* classes

• Each device can generate *events* and send them to other devices or to themselves, as well as receive events from other devices. In the example, these events can be the generation of a packet at a mobile terminal, the beginning or the end of its transmission to the base station, the reception of an acknowledgement, and so on.

• Events are represented by classes which contain information on when the event occurs (the time stamp), which device generated the event (the sender), which device will handle it (the receiver) and optionally any other information to be transmitted from the sender to the receiver.

• Device classes have a method (the event handler) for each different event they are going to receive, and this method is called (by the scheduler) whenever the event occur.

• In order to avoid problems with finite numerical precision of the floating point representation of real numbers, in NePSi, the time is represented with the special class “Time” which uses only integer numbers, and provides a large set of methods to deal with synchronous systems like time division multiple access (TDMA) transmission systems, where the time axis is divided into frames and slots.

• Finally, a “debugger” class is defined to help the developer of the simulation in the hard task of verifying that the simulation behaves correctly.
# Speed Benchmarks

<table>
<thead>
<tr>
<th>Model</th>
<th>INeSiS Simulation Time</th>
<th>Memory Usage</th>
<th>Matlab Simulation Time</th>
<th>Memory Usage</th>
<th>Ptolemy Simulation Time</th>
<th>Memory Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rayleigh fading channel&lt;sup&gt;(1)&lt;/sup&gt;</td>
<td>68.4 s</td>
<td>700 Kbyte</td>
<td>1934 s</td>
<td>20 Mbyte</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>CDMA Blind receiver&lt;sup&gt;(2)&lt;/sup&gt;</td>
<td>26.10 s</td>
<td>600 Kbyte</td>
<td>876 s</td>
<td>20 Mbyte</td>
<td>576.5 s</td>
<td>12 Mbyte</td>
</tr>
<tr>
<td>M/M/1 queue&lt;sup&gt;(3)&lt;/sup&gt;</td>
<td>60.5 s</td>
<td>1 Mbyte</td>
<td>-</td>
<td>-</td>
<td>931 s</td>
<td>15 Mbyte</td>
</tr>
<tr>
<td>PRMA System&lt;sup&gt;(4)&lt;/sup&gt;</td>
<td>412 s</td>
<td>1.6 Mbyte</td>
<td>-</td>
<td>-</td>
<td>19360 s</td>
<td>19 Mbyte</td>
</tr>
</tbody>
</table>

(1) 6 taps GSM Rayleigh fading channel sampled at 4.096 Msamples/s simulated for 1000000 chips


(3) M/M/1 queuing system with arrival rate of 1 arrival per second and mean service time of 0.7 s simulated for 1000000 s.

(4) Voice-Date PRMA system (S. Nanda, IEEE Transaction on Communications, vol. 42, May 1994) with 20 voice terminals, 20 data terminals and data arrival rate of 10 pkt/s, simulated for 10000 s (12.5 millions of slots)

All presented results have been obtained with a Pentium Pro 200 MHz with 128 Mbyte RAM running a Linux (INeSiS and Ptolemy) or Windows NT (Matlab) operating system.
MuDiSP3 Satellite Libraries

- S-UMTS Uplink and Downlink
- Corazza-Vatalaro Satellite Channel @ various
  - mobile speeds
  - rates
  - environments
  - elevation angles
- GEO Ka Satellite System (Italsat/HB6)
- MIMO Satellite channels (planned)
- Space Time Coded Transmit diversity for GEO/LEO satellite systems (planned)
NePSing Satellite Systems

- PRMA for LEO/GEO
- DVB-RCS (Resource Management, QoS)
Availability

- GNU License
- MuDiSP3 (maintainer L.S.Ronga)
  - [http://lenst.det.unifi.it/mudisp3](http://lenst.det.unifi.it/mudisp3)
  - libraries will be posted Jan03
- NePSing (maintainer T.Pecorella)
  - email [tommaso.pecorella@cnit.it](mailto:tommaso.pecorella@cnit.it)