

# Simulation in Cyber Security

Andres Ojamaa

Institute of Cybernetics, Tallinn University of Technology

CS Theory Days, 25 Jan 2008, Põlva

# Outline

## Cyber Security and Simulation

- What is this thing called Cyber Security?

- Simulation Needs and Tasks

- Simulation: Advantages, Disadvantages

## Computer Network Simulation Software

- Discrete Event Simulation

- Network Simulator (ns-2)

- OPNET®

- OMNeT++

## Rich Components in CoCoViLa

- Rich Components

- Interactivity in CoCoViLa



# What is this thing called Cyber Security?

- ▶ Information security
  - ▶ Confidentiality, Availability, Integrity
  - ▶ ... also Forensics, Auditing, Disaster recovery, ...
- ▶ Information assurance
- ▶ Cyber security
- ▶ Cyber defence

Laws? Military?



# Simulation Needs and Tasks

- ▶ Education
- ▶ Planning and design of computer networks
- ▶ Application performance analysis
- ▶ Simulation of attacks on attack trees
- ▶ Simulation of worms, viruses
- ▶ Denial of Service, hardware failures
- ▶ Network models
- ▶ Visualization



# Simulation: Motivation

- ▶ Learn by playing "what-if" games
- ▶ Optimize budget
- ▶ Discover design flaws earlier
- ▶ Visualize fast and invisible processes
- ▶ Save resources



# Simulation: Shortcomings

- ▶ Not always reliable
- ▶ Models can be expensive to build and maintain
- ▶ Not trivial to get it right: performance, accuracy, level of abstraction
- ▶ There is no single tool to answer all questions
- ▶ The Internet is constantly changing
- ▶ Hard to get real data, adaptivity of network protocols



# Discrete Event Simulation

Most of the simulators are just class libraries and frameworks to build the simulation program upon.

- ▶ Infrastructure: tracing, events, objects, connections, ...
- ▶ Event queue and queue manager
- ▶ Class libraries of standard components: clocks, protocol implementations, traffic generators
- ▶ Tools for postprocessing and visualizing the traces  
(Network ANimator)



## ns-2

Ns-2 is a discrete event simulator targeted at networking research.

- ▶ Supports: TCP, routing, ...
- ▶ Implemented in C++, Tcl
- ▶ Portable, free software
- ▶ Single threaded, no distributed computations
- ▶ Separate tools for model construction and processing the output





## Using ns-2: hello, world

```
% set ns [new Simulator]
% $ns at 1 "puts \"hello, world\""
% $ns at 2 "exit"
% $ns run
hello, world
```



# Basic ns-2 Simulation

- ▶ Create scheduler
- ▶ Build network and connections
- ▶ Generate traffic
- ▶ Analyze traces



## Ns-2 Script Examples

```
set n0 [$ns node]  
$ns duplex-link $n0 $n1 5Mb 2ms DropTail  
set tcp [$ns create-connection  
    TCP $n0 TCPSink $n1 0]  
set ftp [new Application/FTP]  
$ftp attach-agent $tcp
```



- ▶ Really fancy: integrated software and hardware appliances
- ▶ Graphical user interface
- ▶ Tools for various tasks: design and planning, auditing, monitoring
- ▶ Lots of pre-built components
- ▶ Really expensive



# OMNeT++ [3]

The screenshot displays the OMNeT++ simulation environment with the following components:

- Top Panel:** (cQueue) multicastNetwork.router1.ppp[0].queue. It shows a table of objects:
 

Class	Name	T=0.911
IPDatagram	udpAppData-2	T=0.911
- Left Panel:** (StandardHost) multicastNetwork.host1 (id=2). It shows a network diagram with a blackboard, 1+1 routes, routingTable, tcp, pingApp, networkLayer, and a host connected to a router (172.0.0.1 / 10M). The host is labeled ppp[0] and is sending a (PPPFrame)udpAppData-2.
- Center Panel:** OMNeT++/Tkenv - multicastNetwork. It shows a tree view of the simulation hierarchy:
  - multicastNetwork (MulticastNetwork)
    - parameters (cArray)
    - gates (cArray)
    - host1 (StandardHost)
    - host2 (StandardHost)
    - host3 (StandardHost)
    - host4 (StandardHost)
    - host5 (StandardHost)
    - host6 (StandardHost)
    - router1 (Router)
    - router2 (Router)
    - router3 (Router)
    - router4 (Router)
    - scheduled-events (cArray)
      - pppEndTxEvent
      - pppEndTxEvent
      - nnfEndTxEvent
- Right Panel:** (IPDatagram) ...heduled-eve... It shows the details of the scheduled event:
 

```

short version = 4
short headerLength = 20
IPAddress srcAddress = 172.0.0.2, output port
Event #108, T=0.9014082752 (901ms), Module
Received (IPDatagram)udpAppData-3 for transmissi
Starting transmission of (PPPFrame)udpAppData-3
Event #109, T=0.9014082752 (901ms), Module

```
- Bottom Panel:** (MulticastNetwork) multicastNetwork. It shows a network diagram with four routers (router1, router2, router3, router4) and six hosts (host1, host2, host3, host4, host5, host6). Router1 is highlighted with a red box. Host1 is connected to Router1. Router1 is connected to Router2, Router3, and Router4. Router2 is connected to Router3 and Router4. Router3 is connected to Router4. Host2 is connected to Router1. Host3 is connected to Router1. Host4 is connected to Router3. Host5 is connected to Router2. Host6 is connected to Router4.



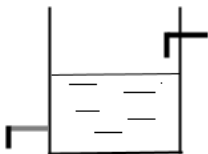
# Rich Components

Rich Components — Concepts of (simulation) domains

- ▶ *Visual representation* for visual programming
- ▶ *Logical part* for abstract properties
- ▶ *Program component* implementing computation algorithms
- ▶ *Daemon part* for interactive properties



# Rich Component: Example



```
1 public class BoilerDaemon
2     implements RunnableDaemon,... {
3     public void run() {
4         ...
5     }
6 }
```

```
1 public class Boiler {
2     /*@ specification Boiler {
3         double inFlow, outFlow;
4         double q, T, minQ, qinit;
5         void done;
6         q = qinit + inFlow - outFlow;
7         q, minQ -> done {test};
8     }@*/
9
10    public void test(double amount, double min) {
11        if (amount < min)
12            beep0;
13    }
14 }
```



# Workflow in CoCoViLa

1. **Scheme**
2. Textual specification
3. Internal representation
4. Proof = Algorithm
5. Java program
6. **Output**





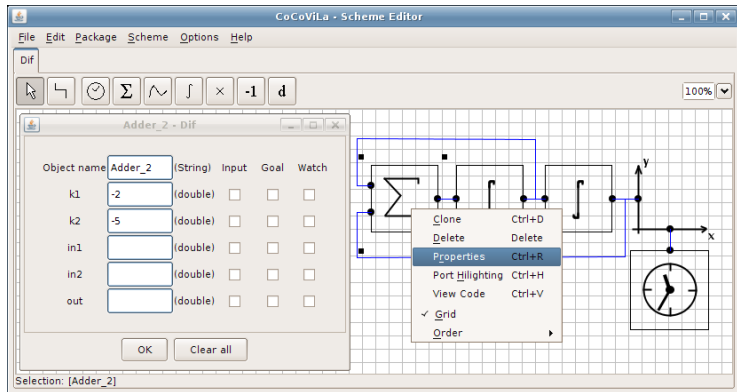
# Interactivity in CoCoViLa Simulations

A programming interface between the generated Java program and the scheme was needed...

```
ProgramContext.getFieldValue(objectName,  
                             fieldName);
```



# Scheme Editor: Oscillator



# Oscillator: Textual Representation and Algorithm

Dif - Specification

Specification Program Run results

Compute goal Compute all

```
1 public class Dif extends Euler {
2   /*@ specification Dif super Euler {
3     Integrator Integrator_2;
4     Integrator_2.T = 5;
5     Integrator_2.initstate = 1;
6     Adder Adder_2;
7     Adder_2.k1 = -2;
8     Adder_2.k2 = -5;
9     Graph Graph_3;
10    Graph_3.seriesName = "Dif";
11    Integrator Integrator_1;
12    Integrator_1.T = 5;
13    Integrator_1.initstate = 0;
14    Clock Clock_3;
15    Clock_3.T = 5;
16    Clock_3.initstate = 0;
17    time = 200;
18    Integrator_2.in = Adder_2.out;
19    Integrator_1.in = Integrator_2.out;
20    Adder_2.in1 = Integrator_2.out;
21    Integrator_1.out = Adder_2.in2;
22    Graph_3.y = Integrator_1.out;
23    Clock_3.out = Graph_3.x;
24  }@*/
25 }
```

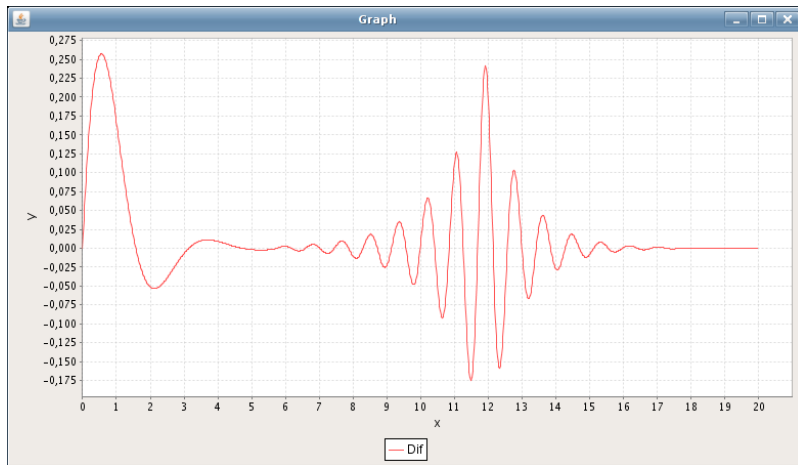
Algorithm Visualizer

Dif

```
spec : Integrator_2.initstate = 1
spec : Adder_2.k1 = -2
spec : Integrator_2.T = 5
Clock_3 : initstate = 0
spec : Integrator_1.T = 5
spec : finalstate_name = "finalstate"
spec : Integrator_1.initstate = 0
spec : initstate_name = "initstate"
spec : Adder_2.k2 = -5
spec : Graph_3.seriesName = "Dif"
spec : time = 200
spec : Clock_3.T = 5
spec : Clock_3.initstate = 0
Graph_3 : seriesName -> init_ready{init}
spec : alias (double) initstate = (*.initstate)
spec : initstate_name, initstate -> done_print_initstate {print_state}
spec : [ state -> nextstate], initstate, time -> finalstate {proc run}
      |Subtask [ state -> nextstate]:
      |Integrator_1 : out = state
      |Clock_3 : nextstate = state + 1/T
      |Clock_3 : out = state
      |Integrator_2 : out = state
      |spec : Clock_3.out = Graph_3.x
      |spec : Graph_3.y = Integrator_1.out
      |spec : Integrator_1.out = Adder_2.in2
      |spec : Adder_2.in1 = Integrator_2.out
      |spec : Integrator_1.in = Integrator_2.out
      |Integrator_1 : nextstate = state + in / T
      |Adder_2 : out = k1 * in1 + k2 * in2
      |Graph_3 : x, y -> drawing_ready{draw}
      |spec : Integrator_2.in = Adder_2.out
      |Integrator_2 : nextstate = state + in / T
      |spec : alias (double) nextstate = (*.nextstate)
      |end of: spec : [ state -> nextstate], initstate, time -> finalstate {proc_run}
      |spec : finalstate_name, finalstate -> done_print_finalstate {print_state}
```



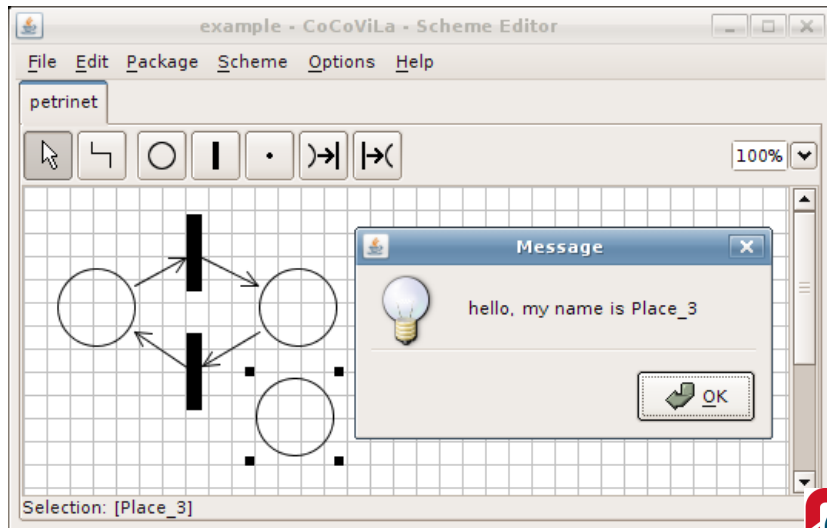
# Oscillator: Simulation Result



Fields  $k_1$  and  $k_2$  of the Adder were changed.



# Interactive Rich Component



# Future Work

- ▶ Develop an expert system shell for CoCoViLa
- ▶ Implement a simple simulation engine as a rich component
- ▶ Build an AS level model of the Estonian Internet
- ▶ Collect and accumulate expert knowledge and real data
- ▶ Experiments with real data
- ▶ Implement interfaces to other simulation packages



# Summary



# Thank you for your attention!

Supporters:

- ▶ Institute of Cybernetics
- ▶ Estonian Information Technology Foundation
- ▶ Tiger University





# References

- ▶ CoCoViLa — Compiler Compiler for Visual Languages,  
<http://www.cs.ioc.ee/~cocovila/>
- ▶ ns-2 — Network Simulator,  
<http://www.isi.edu/nsnam/ns/>
- ▶ OMNeT++ — Discrete event simulator,  
<http://www.omnetpp.org/>
- ▶ OPNET®, <http://www.opnet.com/>

