A Graphical Development and Simulation Environment for TinyOS-based Wireless Sensor Networks

http://ptolemy.eecs.berkeley.edu/viptos

Elaine Cheong   Edward A. Lee   Yang Zhao   Christopher Brooks

Highlights

Viptos (Visual Ptolemy and TinyOS) is an integrated graphical development and simulation environment for TinyOS-based wireless sensor networks.

Graphical Development. Viptos allows developers to create block and arrow diagrams to construct TinyOS programs from any standard library of nesC/TinyOS components.

Code Generation. The tool automatically transforms the diagram into a nesC program that can be compiled and downloaded from within the graphical environment onto any TinyOS-supported target hardware.

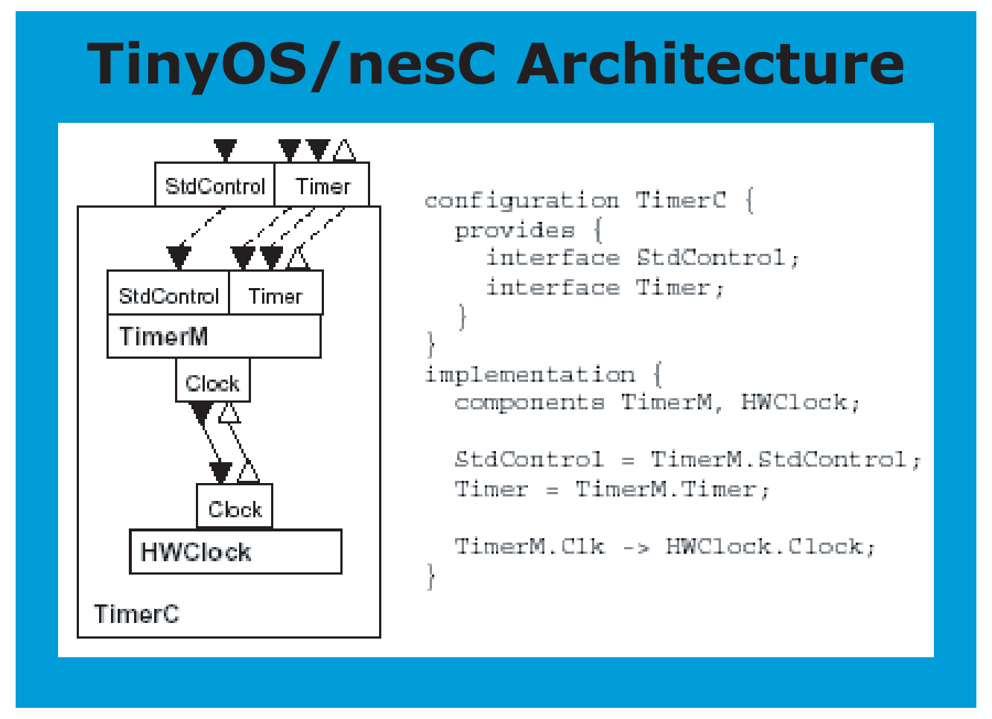
Simulation. Viptos provides interrupt-level simulation of actual TinyOS programs, with packet-level simulation of the network, while allowing the developer to use other models of computation available in Ptolemy II for modeling various parts of the system.

Background

Viptos is built upon TOSSIM and the Ptolemy II VisualSense environment for network-level modeling and simulation of wireless sensor networks.

TOSSIM is a text-based interrupt-level simulator for TinyOS programs. It runs actual TinyOS code but provides software replacements for the simulated hardware, and models network interaction at the bit or packet level.

TinyOS/nesC Architecture



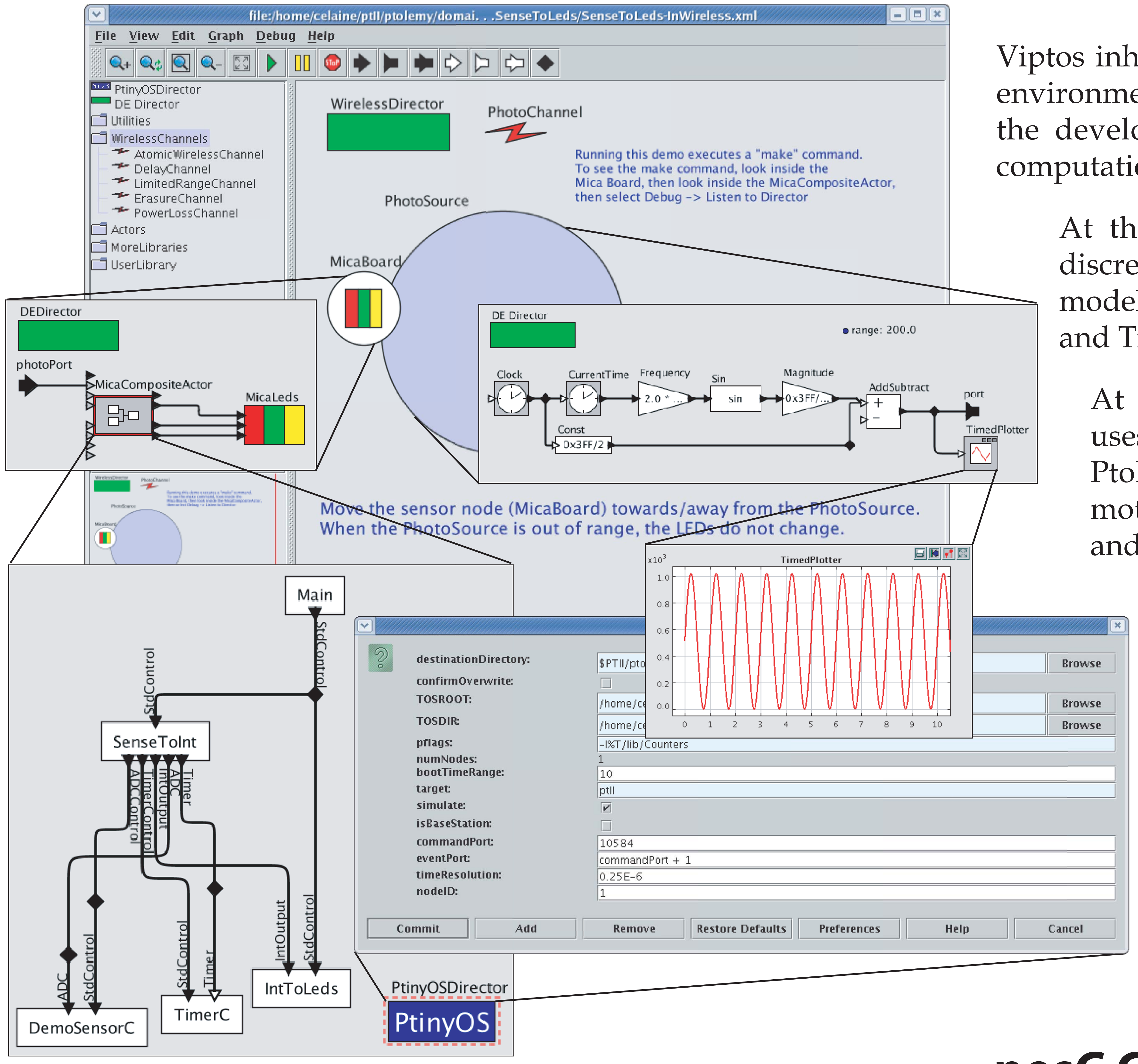
```
configuration TimerC {
  provides {
    interface StdControl;
    interface Timer;
  }
}
implementation {
  components TimerM, HWClock;

  StdControl = TimerM.StdControl;
  Timer = TimerM.Timer;

  TimerM.Clk -> HWClock.Clock;
}
```

Ptolemy II is a graphical software system for modeling, simulation, and design of concurrent, real-time, embedded systems. Ptolemy II focuses on assembly of concurrent components with well-defined models of computation that govern the interaction between components.

Example: Display Incoming Light Level on LEDs



Running this demo executes a "make" command.
To see the make command, look inside the
Mica Board, then look inside the MicaCompositeActor,
then select Debug -> Listen to Director

Move the sensor node (MicaBoard) towards/away from the PhotoSource.
When the PhotoSource is out of range, the LEDs do not change.

destinationDirectory:	$PTII/pto
confirmOverwrite:	□
TOSROOT:	/home/ce
TOSDIR:	/home/ce
pflags:	-I%T/lib/Counters
numNodes:	1
bootTimeRange:	10
target:	ptII
simulate:	☑
isBaseStation:	□
commandPort:	10584
eventPort:	commandPort + 1
timeResolution:	0.25E-6
nodeID:	1

Simulation

Viptos inherits the actor-oriented modeling environment of Ptolemy II, which allows the developer to use different models of computation at each level of simulation.

At the lowest level, Viptos uses the discrete-event scheduler of TOSSIM to model the interaction between the CPU and TinyOS code that runs on it.

At the next highest level, Viptos uses the discrete-event scheduler of Ptolemy II to model interaction with mote hardware, such as the radio and sensors.

This level is then embedded within VisualSense to allow modeling of the wireless channels to simulate packet loss, corruption, delay, etc.

The user can also model and simulate other aspects of the physical environment including those detected by the sensors (e.g., light, temperature, etc.), terrain, etc.

Viptos Harvest Tools

The Viptos utilities, *nc2moml* and *ncapp2moml*, harvest existing TinyOS components and applications and convert them into a format that can be displayed as block (and arrow) diagrams and simulated. These harvest tools use output from the nesC 1.2 compiler.

nesC Code Generation

The PtinyOSDirector embedded in each node can be configured to generate nesC code for simulation or for loading onto the target hardware.

```
configuration _SenseToLeds_InWireless_MicaBoard_MicaActor3198 {
}
implementation {
components Main, TimerC, IntToLeds, SenseToInt, DemoSensorC;
SenseToInt.TimerControl -> TimerC.StdControl;
SenseToInt.Timer -> TimerC.Timer[unique("Timer")];
SenseToInt.IntOutput -> IntToLeds.IntOutput;
Main.StdControl -> IntToLeds.StdControl;
Main.StdControl -> SenseToInt.StdControl;
SenseToInt.ADC -> DemoSensorC.ADC;
SenseToInt.ADCControl -> DemoSensorC.StdControl;
}
```

Chess
University of California, Berkeley
Center for Hybrid & Embedded Software Systems