PtinyOS: Simulating TinyOS in Ptolemy II

Elaine Cheong
celaine@eecs.berkeley.edu

EECS 290N Project Paper
December 17, 2004

Abstract

TinyOS is a component-based operating system designed for wireless embedded sensor networks. TOSSIM provides discrete event simulation of homogeneous TinyOS programs at the interrupt level. This paper presents PtinyOS, an integrated development environment built in Ptolemy II for developing and simulating heterogeneous TinyOS programs. PtinyOS will allow application developers to easily transition between high-level simulation of algorithms to low-level implementation and simulation. This paper also presents the semantics of the integration between the Ptolemy II and TinyOS/TOSSIM programming and execution models.

1 Introduction

Wireless sensor networks provide a way to create flexible, tetherless, automated data collection and monitoring systems. Building sensor networks today requires piecing together a variety of hardware and software components, each with different design methodologies and tools, making it a challenging and error-prone process today. Typical networked embedded system software development may require the design and implementation of device drivers, network stack protocols, scheduler services, application-level tasks, and partitioning of tasks across multiple nodes. Little or no integration exists among the tools necessary to create these software components, mostly because the interactions between the programming models are poorly understood. In addition, the tools typically have little infrastructure for building models and interactions that are not part of their original scope or software design paradigms. The goal of this work is to create integrated tools for networked embedded application developers to model and simulate their algorithms and quickly transition to testing their software on real hardware in the field, while allowing them to use the programming model most appropriate for each part of the system.

Battery-operated nodes can preserve energy by entering sleep mode when no interesting events are happening. In this type of execution, events drive the behavior of the system. TinyOS [7] is an event-driven, component-based runtime environment for the Berkeley/Crossbow sensor network nodes known as *motes*. TinyOS components are written in an object-oriented style using nesC [4], an extension to the C programming language. TOSSIM [8] is a TinyOS simulator for the PC that can execute nesC programs designed for a mote. It contains a discrete event simulation engine which allows modeling of various hardware and other interrupt events. Although a TinyOS program consists of a graph of mostly pre-existing components, users must write their programs in a multi-file, text-based format, whereas a graphical block diagram programming environment would be much more intuitive. Additionally, TOSSIM is not designed to simulate a heterogeneous architecture; the same nesC code must be run on every simulated mote.

Ptolemy II [2] is a software system for modeling, designing, and simulating heterogeneous, concurrent, real-time, embedded systems. Application developers can design and model various types of systems in Ptolemy II by choosing the execution semantics (called a *domain*) that best fits their particular type of application domain. Ptolemy II supports hierarchical nesting of heterogeneous models of computation. Ptolemy II includes VisualSense, a network-level modeling environment for sensor networks based on the discrete event (DE) domain. However, there is currently no mechanism for transitioning from a sensor network application developed in VisualSense to an implementation for real hardware without rewriting the code from scratch for the target platform.

Integrating TinyOS/TOSSIM and Ptolemy II combines the best of both worlds. TinyOS/TOSSIM provides a platform that works on real hardware with a library of components that implement low-level routines. Ptolemy II provides a graphical modeling environment that supports heterogeneous systems. This paper describes the integration of the programming and execution models and the component libraries of these two systems, which is necessary for building an integrated tool chain for designing, simulating, and deploying sensor network applications.

Section 2 describes the TinyOS and TOSSIM programming models. Section 3 describes the architecture of the integrated Ptolemy II and TinyOS toolchain and investigates the semantics of this interface. Sections 4 and 5 conclude with a discussion of related and future work.

2 TinyOS/TOSSIM Programming Models

A TinyOS program consists of a set of nesC components that are “wired” together. Figure 1a shows a program called SenseToLeds that displays the value of a photosensor in binary on the LEDs. TinyOS includes a library of nesC components, including the ones listed in SenseToLeds, such as Main, SenseToInt (shown in Figure 1b), IntToLeds, TimerC, and Sensor.

A component exposes a set of interface methods. The component implements its `provides` methods and expects another component to implement its `uses` methods. In nesC, interfaces can also be parameterized to provide multiple instances of the same interface in a single component. In Figure 1a, the TimerC.Timer interface is parameterized. The Timer interface of SenseToInt connects to a unique instance of the corresponding interface of TimerC. If another component connects to the TimerC.Timer interface, it would be connected to a different instance. Each timer could be initialized with different periods.

In TinyOS, there is a single thread of control managed by the scheduler, which may be interrupted by hardware events. Component methods encapsulate hardware interrupt handlers. Methods may transfer the flow of control to another component by calling a `uses` method. Computation performed in a sequence of method calls must be short, or it may block the processing of other events. A long running computation can be encapsulated in a *task*, which a method *posts* to the scheduler task queue. The TinyOS scheduler processes the tasks in the queue in FIFO order whenever it is not executing an interrupt handler. Tasks are atomic with respect to other tasks and do not preempt other tasks.

When a user compiles a TinyOS program for a mote, the nesC compiler automatically searches the TinyOS component library paths for included components, including directories containing the components that encapsulate the hardware components specific to the mote platform, such as the clock, radio, and sensors. The nesC compiler generates a pre-processed C file, which can then be sent to the `gcc` cross compiler for the specific type of mote.

TinyOS programs can also be compiled for simulation on a PC. In this case, the nesC compiler follows the same procedure but replaces the TinyOS scheduler and device drivers with TOSSIM code. TOSSIM allows one or more nodes with the same TinyOS program to be simulated by maintaining a copy of each component state for each simulated node. Support for these copies is built into the nesC compiler so that the user does not need to modify the TinyOS program source code.

The TOSSIM scheduler contains a task queue similar to the regular TinyOS scheduler. However, the TOSSIM scheduler also contains an ordered event queue. Events in this queue have a timestamp implemented as a

```
configuration SenseToLeds {
} implementation {
  components Main, SenseToInt, IntToLeds,
      TimerC, DemoSensorC as Sensor;

  Main.StdControl -> SenseToInt;
  Main.StdControl -> IntToLeds;

  SenseToInt.Timer ->
      TimerC.Timer[unique("Timer")];
  SenseToInt.TimerControl -> TimerC;
  SenseToInt.ADC -> Sensor;
  SenseToInt.ADCControl -> Sensor;
  SenseToInt.IntOutput -> IntToLeds;
}
```

```
module SenseToInt {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface StdControl
        as TimerControl;
    interface ADC;
    interface StdControl
        as ADCControl;
    interface IntOutput;
  }
} implementation {
    ...
}
```

Figure 1. Sample nesC source code.

`long long` (a 64-bit integer on most systems). The smallest time resolution is equal to 1 / 4MHz, the original CPU frequency of the Rene/Mica motes. In the main scheduling loop, the TOSSIM scheduler first processes all tasks in the task queue in FIFO order. If there is an event in the event queue, it then updates the simulated time to the timestamp of the new event and processes the event. Upon initialization, TOSSIM inserts a boot up event into the event queue. The processing of an event may cause tasks to be posted to the task queue and creation of new events with time stamps possibly equal to the current time stamp.

3 PtinyOS

PtinyOS is the integration of TinyOS and Ptolemy II and allows for graphical development of a heterogeneous set of TinyOS programs. *nc2momllib* is a pre-runtime tool that converts nesC files in the TinyOS component library into the MoML (Modeling Markup Language) format required by Ptolemy II. Ptolemy II uses this XML-based format to display TinyOS components as graphical blocks. The *PtinyOS Director* is a runtime tool that contains facilities for code generation, simulation, and target code deployment.[1]

[1]The initial versions of the nesC visualization and code generation facilities in Ptolemy II were written by Yang Zhao and Edward A. Lee.

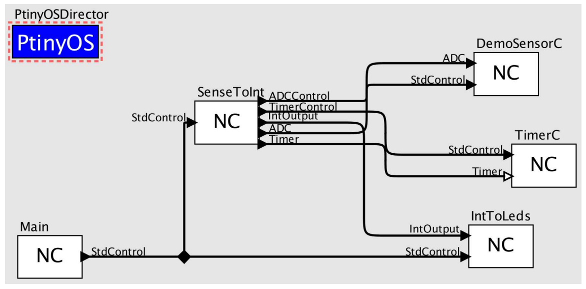

Figure 2. SenseToLeds in Ptolemy II.

Static semantics In Ptolemy II, basic executable code blocks are called actors and may contain input and output ports. An input port may be a simple port that allows only a single connection, or it may be a *multiport* that allows multiple connections. The code block is stored in a class, and an actor is an instance of the class. PtinyOS is built in Ptolemy II and inherits these semantics.

In PtinyOS, nesC components are represented by classes, and component interfaces and interface methods are represented by ports. `uses` methods are represented by output ports, and `provides` methods are represented by input ports. Parameterized interfaces in nesC with a single index are represented by multiports. For multiple connections to a simple *provides* interface, each caller will call the same callee. For multiple connections to a *provides* parameterized interface, each caller will call its own copy of the callee. The semantics for a *uses* parameterized interface are not defined, since they are neither well-defined nor used in TinyOS. PtinyOS does not support multiply indexed parameterized interfaces (it is not used in any existing TinyOS components).

nesC components are converted into Ptolemy II classes via *nc2momllib*, which I have implemented as an extension to the nesC compiler. For each valid component file in the TinyOS library, *nc2momllib* generates an XML file containing MoML syntax that specifies the name of the component, as well as the name and input or output direction of each port, and whether it is a multiport.

Figure 2 shows a graphical representation in Ptolemy II of some of the classes created by *nc2momllib*. The figure shows the equivalent wiring diagram for the SenseToLeds configuration shown in Figure 1a. Note that the TimerC component contains a parameterized interface, or multiport, as indicated by the white triangle. A simple interface, or simple port, is represented by a black triangle.

Runtime semantics The PtinyOS Director controls code generation and compilation, simulation, and target code deployment for a single node. Running the model shown in Figure 2 causes the Director to generate a nesC component file for SenseToLeds, equivalent to that shown in Figure 1a. It then uses the nesC compiler to generate a C file containing modified TOSSIM scheduler functions that can be compiled into a shared library and loaded into Ptolemy II and run via JNI method calls. The Director generates a makefile to perform the compilation, as well as a Java class to perform the loading. The PtinyOS Director allows the user to easily change a compilation parameter to compile and deploy code for a specific mote target from within Ptolemy II.

All TOSSIM components call the `queue_insert_event()` function to insert new events into the TOSSIM event queue. When simulating with PtinyOS, this function also makes a call to Ptolemy II to insert equivalent events into the Ptolemy II event queue using `fireAt()` with the TOSSIM system time as the argument. At each event timestamp, Ptolemy II will call the PtinyOS version of the TOSSIM scheduler to process the event.

The PtinyOS version of the TOSSIM scheduler updates the TOSSIM system time, processes an event in the TOSSIM event queue, and then processes all tasks in the task queue. If the TOSSIM event queue contains another event with the current TOSSIM system time, the scheduler processes the event along with any tasks that may have been generated. This last step is repeated until there are no other events with the current TOSSIM system time. Note that the order in the main loop is the opposite that of the normal TOSSIM, which processes all tasks before updating the time and processing an event in the TOSSIM event queue. This change is required in order to guarantee causal execution in PtinyOS, since tasks may generate events with the current TOSSIM time stamp. Otherwise, new events may have a time stamp that is before the current Ptolemy II system time.

The PtinyOS Director and the nesC components for the program graph can be embedded in a PtinyOSActor. The PtinyOSActor can be embedded in the DE domain of Ptolemy II. The physical environment can be simulated in DE and the data can be fed to the simulated mote through the PtinyOSActor interface, which serves to convert Ptolemy II token data types into nesC/TOSSIM data types. For example, most actors used in the DE domain of Ptolemy II communicate via tokens with values of type double. The ADC channel of a mote uses 10-bit unsigned values. The PtinyOSActor automatically performs the lossy conversion from double to an unsigned integer value that is masked for 10-bit usage. The PtinyOSActor also performs conversions from a char representing a LED or radio signal value in TOSSIM into a boolean-valued token for Ptolemy II.

By embedding multiple PtinyOSActors, each controlled by a different PtinyOS Director, in the discrete event (DE) domain of Ptolemy II, multiple nodes with different pro-

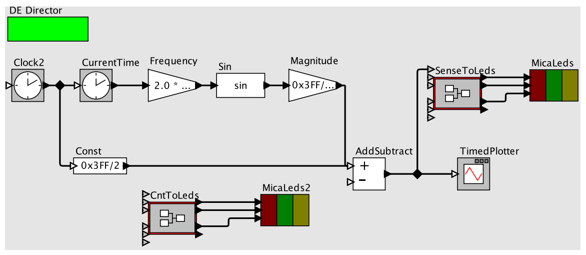

Figure 3. Multiple nodes in Ptolemy II.

grams can be simulated at the same time. Separately compiled and loaded shared libraries prevent name space collision between different simulated TinyOS programs. Figure 3 shows a simulation of two nodes, one containing the SenseToLeds program, and other containing the CntToLeds program which displays an increasing count in binary on the LEDs.

4 Related Work

GRATIS II [1] is a Graphical Development Environment for TinyOS built on top of GME 3 (Generic Modeling Environment). The TinyOS component library is available as graphical blocks with GRATIS II. Given a valid model, the GRATIS II code generator can transform all the interface and wiring information into a set of nesC target files. However, GRATIS II was developed mainly for static analysis of TinyOS component graphs and does not support simulation.

TinyViz [8] is a Java-based graphical user interface for TOSSIM. Other simulators used in the TinyOS community for cycle accurate simulation/emulation of the Atmel AVR (processor used in the motes) instruction set include ATEMU [9] and Avrora [10]. However, these tools do not support graphical development of TinyOS component graphs and do not support heterogeneous nodes.

Em* [5] is toolsuite for developing sensor network applications on Linux-based hardware platforms called microservers. It supports deployment, simulation, emulation, and visualization of live systems, both real and simulated. EmTOS [6] is an extension to Em* that enables an entire nesC/TinyOS application to run as a single module in an Em* system. The EmTOS wrapper library is similar to the TOSSIM simulated device library. Em* modules are implemented as user-space processes, which means the minimum granularity of a timer is 10ms, corresponding to the Linux jiffy clock that is part of the scheduler in the Linux 2.4 kernel. Thus, EmTOS modules are restricted to using the Linux scheduler as the main programming model.

5 Conclusion and Future Work

This paper describes the integration of TinyOS and Ptolemy II to form a seamless toolchain for developing, simulating, and deploying sensor network applications on the motes. It describes methods for parsing and generating nesC files and also discusses how the programming models of TOSSIM and Ptolemy II fit together. The PtinyOS domain currently has facilities for simulating output to the LEDs and radio channel and simulating input to the ADC (analog-to-digital) channels. Radio channel input simulation is still under development.

The radio and ADC channels can easily be integrated with the wireless channels of VisualSense in Ptolemy II. PtinyOS provides a bridge between the network-level modeling and simulation of VisualSense and the bit-level simulation of TinyOS. A user could simulate and test high-level protocols in VisualSense and transition to testing an actual implementation for the motes through simulation in PtinyOS with eventual deployment to the target hardware. Simulation speed and memory requirements could be improved by taking advantage of the TOSSIM ability to simulate multiple homogeneous nodes, which PtinyOS does not use currently.

The PtinyOS environment also provides a good basis for investigating actor-oriented models for sensor node software. TinyOS/nesC component interfaces use call/return semantics. In some applications, a message-based actor-oriented interface may be more suitable. galsC [3] is a extension to nesC that provides an actor-oriented model on top of the regular nesC programming model. galsC is currently targeted for the motes only. An extension to PtinyOS for galsC would provide a good simulation environment. Another interesting area of future research would be integrating Em* with PtinyOS, since Em* allows for simulation with real radios rather than simulated channels and also allows for simulation of heterogeneous architectures (a combination of motes and microservers), rather than the homogeneous motes required by TOSSIM.

References

[1] GRATIS. http://www.isis.vanderbilt.edu/projects/nest/gratis/.

[2] Ptolemy project. http://ptolemy.eecs.berkeley.edu.

[3] E. Cheong and J. Liu. galsC: A language for event-driven embedded systems. In *DATE05 (to appear)*.

[4] D. Gay et al. The nesC language: A holistic approach to networked embedded systems. In *PLDI'03*.

[5] L. Girod et al. Emstar: A software environment for developing and deploying wireless sensor networks. In *USENIX 2004 Annual Technical Conference*.

[6] L. Girod et al. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *SenSys'04*.

[7] J. Hill et al. System architecture directions for networked sensors. In *ASPLOS 2000*.

[8] P. Levis et al. TOSSIM: accurate and scalable simulation of entire tinyos applications. In *SenSys'03*.

[9] J. Polley et al. Atemu: A fine-grained sensor network simulator. In *SECON'04*.

[10] B. Titzer et al. Avrora: Scalable sensor network simulation with precise timing. online manuscript, 2004.