Motes, nesC, and TinyOS

Gary Wong

December 9, 2003
Introduction

• System overview

• Mote hardware

• nesC language

• TinyOS operating system
System overview

Consider an environment which requires fully autonomous operation:

- no mains power

- no wired communication

- no human intervention

What do these limitations make our computers and programs look like?
Mote hardware

- Processor: Atmel AVR ATmega128L µcontroller
  - 128KB flash ROM, 4KB RAM, 4KB E²PROM
  - up to 8MHz
- Radio: Chipcon CC1000
  - UHF transceiver (300MHz–1GHz)
  - FSK modulation, up to 76.8kBaud
- Sensor boards
Programming the AVR architecture

- lots of registers (32)
- RISC, load-store model
- conventional stack
- linear Harvard-style address space
- highly orthogonal instruction set

⇒ nice for conventional compilers
nesC language

A dialect of C:

• imperative, very C-like at the low level

• more declarative style at top level

• highly modular

• whole program compilation
nesC language

- Programs are built from components, which are either modules or configurations. Components provide and use interfaces.

- Modules implement interfaces with functions (commands and events); configurations connect interfaces together ("wiring").

- A program always has a top-level configuration.

- The concurrency model is based on tasks and hardware events: tasks never preempt execution, but hardware events do.
The only way to learn a new programming language is by writing programs in it. The first program to write is the same for all languages:

*Print the words*

*hello, world*

— Kernighan and Ritchie, *The C Programming Language (2nd edition)*

But how can we write such a program in an environment with no alphanumeric I/O capability?
module HelloWorldM {
    provides {
        interface StdControl;
    }
    uses {
        interface Timer;
        interface Leds;
    }
}

continues...
implementation {
    command result_t StdControl.init() { ... }
    command result_t StdControl.start() {
        return call Timer.start( TIMER_ONE_SHOT, 1000 );
    }
    command result_t StdControl.stop() { ... }
    event result_t Timer.fired() { ... }
}
nesC example: HelloWorld.nc

configuration HelloWorld {
}
implementation {
    components Main, HelloWorldM, TimerC, LedsC;

    Main.StdControl -> HelloWorldM;
    Main.StdControl -> TimerC;

    HelloWorldM.Timer -> TimerC.Timer[ unique("Timer") ];
    HelloWorldM.Leds -> LedsC;
}
TinyOS operating system

TinyOS is a runtime environment for nesC programs running on Mote hardware:

- Performs some resource management.

- Selected components are linked into program at compile time.

- Written in nesC and C.

- All time-consuming commands are non-blocking.
TinyOS operating system

Provided components include:

- Analogue to digital conversion
- Cryptography
- Data logging
- File system
- I²C communication
- LED control
- Memory allocation
- Random number generation
- Routing
- Sensor board input
- Serial communication (wired and wireless)
- Timers
- Watchdog timer
TOSSIM: Tiny OS SIMulator

- nesC can compile to native binaries.

- The resulting simulator imitates a group of Motes.

- TOSSIM emulates the Mote peripheral hardware.

- Java GUI (TinyViz) connects to the simulator binary over a socket.
Conclusion

- Mote hardware
- nesC language
- TinyOS operating system