Operating System Support for Handling Heterogeneity in Wireless Sensor Networks

Lucas Wanner
Antônio Augusto Fröhlich

Laboratory for Software/Hardware Integration
Federal University of Santa Catarina

guto@lisha.ufsc.br
http://www.lisha.ufsc.br/

September 21, 2005

WSN Hardware

- Basic recipe
  - Microcontroller
  - Low-power radio
  - Battery
  - Just about any sensor you can think of
- Result
  - Myriad of sensing platforms
  - Similar functionality
  - Different implementations

Current WSN Hardware “Incarnations”

Mote					
Type	**Rene**	**Mica2**	**iMote**	**BTnode rev. 3**	**Telos rev. B**
Institution	UCB	UCB	Intel	ETHZ	UCB
Year	2000	2003	2003	2005	2005
CPU					
Microcontroller	AVR	AVR	ARM	AVR	MSP430
Clock	4 MHz	8 MHz	12 MHz	8 MHz	8 MHz
Program Memory	8 KB	128 KB	512 KB	128 KB	60 KB
RAM	0.5 KB	4 KB	64 KB	256 KB	10 KB
Radio					
Type	RFM	Chipcon	Bluetooth	BT/Chipcon	802.15.4
Frequency (MHz)	916	916	2400	2400/916	2400
Rate (Kbps)	10	40	700	700/40	250

WSN Sensors

- Just about anything!
  - Example: Mica sensor boards
    - Temperature
      - YSI 44006 (analog)
      - Panasonic ERT-J1VR103J (analog)
      - Sensirion SHT11 (digital)
      - Intersema MS55ER (digital)
    - Light
      - CdSe Photocell (analog)
      - TAOS TLS2550 (digital)
    - Accelerometer
      - Analog Devices ADXL202JE (analog)
    - Magnetometer
      - Honeywell HMC1002 (analog)

WSN Programming Models

- As regards the level of abstraction
  - Assembly language
  - Programming language with run-time support libraries
  - Operating system
  - Middleware or virtual machine

Assembly

- Myth
  - Real men do it in assembly!
  - Efficient
  - Absolute control over the hardware
  - No dependencies (compiler, OS, etc)
  - Applications for WSN are usually simple
- Reality
  - Error-prone
  - Little room for code reuse
  - Complexity for applications is a limiting factor
  - Compilers can generate efficient code

Programming Language

- Myth
  - Code reuse through simple libraries
  - Complete control over the hardware
    - Inline assembly?
  - Little overhead
- Reality
  - No context-aware abstraction model
  - Portability easily compromised (assembly and run-time support system)
  - For complex (real) applications, run-time support libraries get close to fully-fledged operating systems

Operating System

- Myth
  - Implementation details hidden by API
  - Application portability sustained
  - Complex programming models
  - Excessive overhead
- Reality?
  - Let’s take a deeper look

A Simple Sensing Application

- Continuously check a temperature sensor, forwarding the acquired data through a serial line
- On the traditional Mica2 motes
- Using two traditional OS for WSN
  - TinyOS
    - Component-based architecture
    - NesC (commands, events, similar to HDLs)
    - Task-based scheduling
    - Layered hardware abstraction
  - Mantis
    - POSIX-like API
    - Monolithic Hardware Abstraction Layer (device drivers)

Application on TinyOS

```
configuration SenseToUART {}

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

Application on Mantis

```
static comBuf send_pkt;

void start (void) {
   send_pkt.size=1;
   while(1) {

     dev_read(DEV_MICA2_TEMP,
              &send_pkt.data[0],1);

     com_send(IFACE_SERIAL, &send_pkt);

   }
}
```

Operating System

- Myth
  - Implementation details hidden by API
  - Application portability sustained
  - Complex programming models
  - Excessive overhead
- Reality
  - Exposed implementation details
  - Neither OS provides an interface for dealing with sensors
  - Neither OS sustains application portability
  - Excessive overhead (as we will see soon)

Middleware / Virtual Machine

- Myth
  - Implementation details and OS hidden by API
    - But usually some escape provided
  - Application portability sustained
  - Adequate programming models (high-level services)
  - Acceptable overhead
- Reality
  - Same design flaws of traditional OS
  - Prohibitive cost
    - Does anyone have a VM for a 4K RAM μ-controller?

A Run-time Support System for WSN

- Programming model that is adequate for application programmers (and not system programmers)
- Effective hardware abstraction mechanisms
    - High-level sensing subsystem
- Application portability
- Little overhead
- ...
- An Application-Oriented Operating System

The EPOS System

- Embedded Parallel Operating System
  - A collection of software components designed according to AOSD principles
  - A meta-programmed framework
  - A set of tools to assist the selection, configuration and adaptation of those software components
- Portability
  - EPOS abstractions interact with hardware components through mediators
  - Hardware mediators sustain an interface contract between system abstractions and the machine

EPOS Sensing Subsystem

- **Sensor** hardware mediators
  - Uniform interface
  - Platform-dependent implementations
  - Example
    - Panasonic ERT-J1VR103J (temperature)
    - Analog Devices ADXL202JE (acceleration)
    - Honeywell HMC1002 (magnetic force)
- **Sentient** system abstractions
  - User-visible software components
  - Platform-independent implementations
  - Example
    - Thermometer
    - Accelerometer
    - Magnetometer

EPOS Sentients

- A **Sentient** abstraction can autonomously react to **Sensor** variations
  - Logs
  - Alarms
  - **Actuator** invocations
- Applications using sentient abstractions can be transparently ported to different sensing platforms
  - **Distinct Sensors**
  - **Same Hardware Mediator interface**

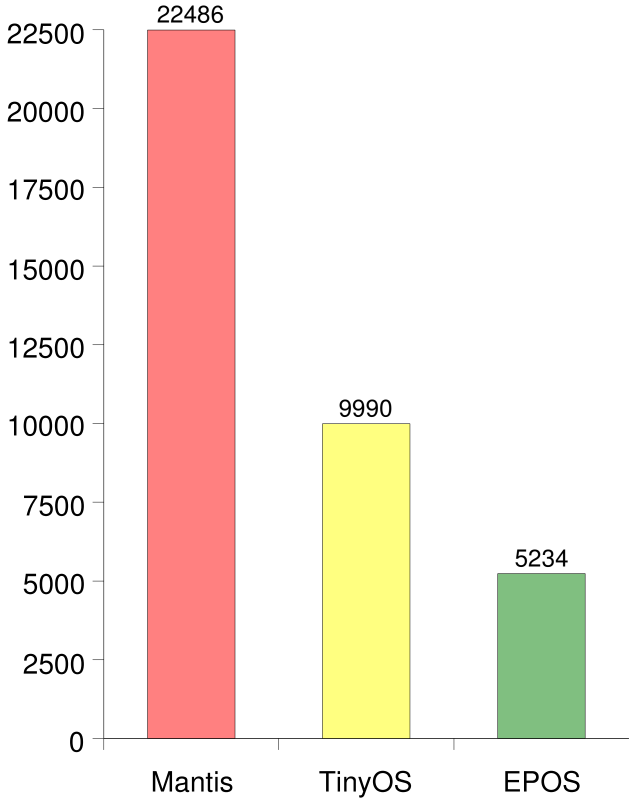
Federal University of Santa Catarina
Laboratory for Software/Hardware Integration

What about the Sensing Application on EPOS?

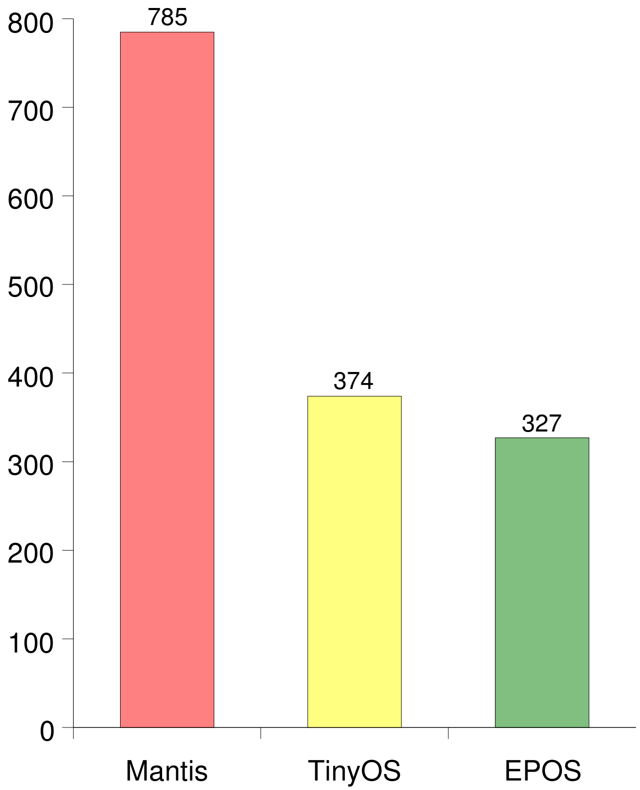
```
#include <thermometer.h>
#include <uart.h>

Thermometer thermometer;    // Sentient
UART uart(9600,8,0,1);      // Mediator

int main()
{
  while(1)
    uart.put(thermometer.get());
}
```

Results: System Footprint

- Build for Mica2 motes (size in bytes, system + application)



Program Memory



Data Memory

Further Work

- Support for different hardware platforms
  - Mica2
  - Telos
  - BT-node
- More Sentient abstractions
  - Integration with EPOS active object model
- Extended evaluation
  - Performance
  - Energy consumption

Conclusions

- We believe we have defined a model for a real WSN run-time support systems
  - Application-oriented system design
  - Sensors
  - Sentients
  - OS relieves applications from the dirty work
- But it is too early to claim success
  - Can we really abstract a considerable set of sensors under a common interface?
    - I.e., is there a temperature sensor common interface?
  - Will sentients fit in tiny platforms?
    - Wireless communication usually demands a lot of resources