Ubiquitous Computing:

Software for WSNs: nesC, TinyOS and software tools

Doina Bucur
doinabucur@incas3.eu
http://www.daimi.au.dk/~doina/
(continued from the lecture on WSNs)

(4) Software for WSNs: Programming, Operating Systems and Tools

We look at the **programming paradigms** for sensor nodes and introduce a mainstream **operating system**, TinyOS.

Then, we list examples of useful **software tools**: a code simulator and a tool for programming over the network.
We look at:

<table>
<thead>
<tr>
<th>René</th>
<th>ATMEL8535</th>
<th>916 MHz radio with bandwidth of 10 kbit/s</th>
<th>512 bytes RAM</th>
<th>8K Flash</th>
<th>TinyOS Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>SenseNode</td>
<td>MSP430F1611</td>
<td>Chipcon CC2420</td>
<td>10K RAM</td>
<td>48K Flash</td>
<td>GenOS and TinyOS Support</td>
</tr>
<tr>
<td>Shimmer</td>
<td>MSP430F1611</td>
<td>802.15.4 Shimmer SR7 (TI CC2420)</td>
<td>48 KB Flash</td>
<td>2 GB microSD Card</td>
<td>TinyOS Support. Built in 3 Axis Accel, Tilt/Vib Sensor. Full range of expansion modules.</td>
</tr>
<tr>
<td>SunSPOT</td>
<td>ARM 920T</td>
<td>802.15.4</td>
<td>512K RAM</td>
<td>4 MB Flash</td>
<td>Java</td>
</tr>
<tr>
<td>Telos</td>
<td>MSP430</td>
<td></td>
<td>2K RAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TelosB</td>
<td>Texas Instruments MSP430 microcontroller</td>
<td>250 kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transceiver</td>
<td>10k RAM</td>
<td>48k Flash</td>
<td>Contiki, TinyOS, SOS and MantisOS Support</td>
</tr>
<tr>
<td>TinyNode</td>
<td>Texas Instruments MSP430 microcontroller</td>
<td>Semtech SX1211</td>
<td>8K RAM</td>
<td>512K Flash</td>
<td>C Programming</td>
</tr>
<tr>
<td>T-Mote Sky</td>
<td>Texas Instruments MSP430 microcontroller</td>
<td>250 kbit/s 2.4 GHz IEEE 802.15.4 Chipcon Wireless Transceiver</td>
<td>10k RAM</td>
<td>48k Flash</td>
<td>Contiki, TinyOS, SOS and MantisOS Support</td>
</tr>
</tbody>
</table>
Why program using an OS?

- Device drivers, scheduler, power management, shared resource access, and network protocols are implemented and tested.
- A programmer need only implement application logic, and “call” existing code.
- The OS programming language is higher-level. nesC: network embedded system C.
- Reliable; large support community. Compatibility with other experiments.

On the other hand:
- Novel programming paradigms
- Alternatives: platform C and/or assembly

http://docs.tinyos.net

Contiki

...many other OSs exist!
WSN programming

... differs from desktop programming: sensor nodes are resource-constrained.

Porting Linux/Windows, and coding in Java, Python, etc, on sensor nodes is rarely feasible. It is feasible on smartphones!

Usual WSN programming:

- sensor code (nesC/TinyOS)
- base station code (nesC/TinyOS)
- gateway code (Java, C, ...)

1.1.1 Anatomy of a Sensor Node (Mote)

Since energy consumption determines sensor node lifetime, sensor nodes, commonly referred to as motes, tend to have very limited computational and communication resources. Instead of a full-fledged, 32-bit or...
static inline void Msp430$IOP$36$IO$toggle()
{
    #line 47
    __nesc_atomic = __nesc_atomic_start();
    *(volatile uint8_t *)49U ^= 0x01 << 4;
    __nesc_atomic_end(__nesc_atomic);
}

mov    r18, r24
in     r25, 0x3f
cli
mov    r30, r24
eor    r31, r31
subi   r30, 0xF8
sbci   r31, 0xFE
ld     r24, Z
cpi    r24, 0xFF
brne   .+32
TinyOS compiler chain

We first look at writing nesC, then at TinyOS.
nesC

TinyOS itself and (most of) its applications are written in nesC:

- C with extra features
- Basic unit of nesC software: the **component**
- Components connect via **interfaces**, which give access to functions of component
- Connections called wiring

Note: these graphs are generated automatically from the application code by the nesdoc tool, see e.g. [http://www.tinyos.net/tinyos-2.x/doc/nesdoc/telosb/](http://www.tinyos.net/tinyos-2.x/doc/nesdoc/telosb/)
nesC components

Very similar to classes, but static:

- No instantiation at runtime (but at compile-time)

Components use or provide interfaces.

Interfaces contain commands and events (both, functions).

```plaintext
module BlinkC {
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer2;
  uses interface Leds;
  uses interface Boot;
}
```

```plaintext
implementation {
  event void Boot.booted() {
    call Timer0.startPeriodic( 250 );
    call Timer1.startPeriodic( 500 );
    call Timer2.startPeriodic( 1000 );
  }

  event void Timer0.fired() {
    call Leds.led0Toggle();
  }

  event void Timer1.fired() {
    call Leds.led1Toggle();
  }

  event void Timer2.fired() {
    call Leds.led2Toggle();
  }
}
```
module BlinkC
{
    uses interface Timer<TMilli> as Timer0;
    uses interface Timer<TMilli> as Timer1;
    uses interface Timer<TMilli> as Timer2;
    uses interface Leds;
    uses interface Boot;
}

implementation
{
    event void Boot.booted()
    {
        call Timer0.startPeriodic( 250 );
        call Timer1.startPeriodic( 500 );
        call Timer2.startPeriodic( 1000 );
    }

    event void Timer0.fired()
    {
        call Leds.led0Toggle();
    }

    event void Timer1.fired()
    {
        call Leds.led1Toggle();
    }

    event void Timer2.fired()
    {
        call Leds.led2Toggle();
    }
}

Zooming in on the Timer interface:

calls command startPeriodic()
signals event fired()

also: replace Timer with any sensor/transceiver operation!
bash-3.2$ make telosb

mkdir -p build/telosb

compiling BlinkAppC to a telosb binary

ncc -o build/telosb/main.exe -fnesc-separator=__ -Wall
-target=telosb -fnesc-cfile=build/telosb/app.c -
DIDENT_APPNAME="BlinkAppC" -fnesc-dump=wiring [..]
BlinkAppC.nc -lm

compiled BlinkAppC to build/telosb/main.exe

2648 bytes in ROM

54 bytes in RAM

bash-3.2$ make telosb install,0 bsl,/dev/ttyUSB0
Event-based programming

- Also called asynchronous programming, or split-phase programming.
- No function call is blocking, so there is no busy-waiting.
  - Efficient, in both memory and time. Suited for real-time processing.
  - A well-known paradigm in operating systems.
- Careful: event handlers can interrupt each other! They should be very brief.
...with **deferred tasks**

```c
/* We completed the acceleration read. Post the task that checks for theft. We defer this somewhat cpu-intensive computation to avoid having the current event run for too long. */

```event` void ReadStream.readDone(error_t ok,[..]) {
    if (ok == SUCCESS)
        post checkAcceleration();
    else
        errorLed();
}
```
Recent: TinyOS also with plain-C threads

TOSThreads.

See apps/tostthreads in the TinyOS installation folder for sample applications.

```c
// Initialize variables associated with each thread
tosthread_t blink0;
tosthread_t blink1;
tosthread_t blink2;

void blink0_thread(void* arg);
void blink1_thread(void* arg);
void blink2_thread(void* arg);

void tosthread_main(void* arg) {
    // Use stack estimator to calculate maximum
    // stack size on a thread by thread basis
    tosthread_create(&blink0, blink0_thread, [...]);
    tosthread_create(&blink1, blink1_thread, [...]);
    tosthread_create(&blink2, blink2_thread, [...]);
}

void blink0_thread(void* arg) {
    for(;;) {
        led0Toggle();
        tosthread_sleep(200);
    }
}

[...]
```
TOSSIM: a TinyOS simulator
Deluge: programming over-the-air

http://docs.tinyos.net/tinywiki/index.php/Deluge_T2
Resources

TinyOS has an **apps** folder: many sample applications

TinyOS book: *TinyOS Programming* by Philip Levis, David Gay

TinyOS website: [http://www.tinyos.net](http://www.tinyos.net)