CS434/534: Topics in Networked (Networking) Systems

WSN/Mobile Systems

Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson
Email: yry@cs.yale.edu

http://zoo.cs.yale.edu/classes/cs434/
Project planning

- Start: Apr. 5
  - A meeting w/ instructor; Due: team, potential topics
- Checkpoint I: Apr. 13 (updated; Thursday)
  - Initial ideas; survey of related work; feedback from instructor/TF
- Checkpoint II: Apr. 20 (updated)
  - Initial design; feedback from instructor/TF
- Checkpoint III: Apr. 27 (updated)
  - Design refined; key components details
- Final report: May 4
  - No more than 6 page write up
Recap: Software Defined Radio

- Instead of hardware-centric radio, making wireless networking systems software-centric.

- Key challenges
  - Handle large data volume
  - Provide hard deadline and accurate timing control
  - Provide easy to use, compose signal processing blocks
Recap: GNU Radio
Software/Execution Model

- **Software model**
  - **C++**
    - Composite pattern to build a hierarchy of blocks
  - **Python**
    - Application management (e.g., GUI)
    - Flow graph construction
    - Non-streaming code (e.g., MAC-layer)

- **Execution model**
  - Python thread for each top_block
Recap: Sora Software Radio

- Efficient impl of blocks using LUT/SIMD
- Utilizing multi-core for streaming processing
- Dedicated core for real-time support
Outline

- Admin and recap
- Wireless and mobile software framework
  - GNURadio
  - SORA
  - TinyOS
Recap: Settings and Requirements of TinyOS

- **Flexible configuration of attached devices**
- **Small foot print**
  - Devices have limited memory and power resources

1.5” x 1.5”
TinyOS: Software Concept

- TinyOS: Generate customized OS + application for each given scenario
  - support one application at a time but flexible reprogramming
Schematic Diagram
TinyOS: Software Concepts

- A TinyOS consists of one or more components linked together
  - software components motivated by hardware component

- Each component specifies that
  - it provides some interfaces
    - allows other components to control it
  - also uses some interfaces
    - control other components
An interface declares
- a set of functions called commands that the provider must implement
- another set of functions called events that the interface user must implement

A uses interfaces I1 and I2

B provides I1

C provides I2

C provides I3
## Interface: Examples

### StdControl.nc

```cpp
interface StdControl {
    command result_t init();
    command result_t start();
    command result_t stop();
}
```

### Timer.nc

```cpp
interface Timer {
    command result_t start(char type, uint32_t interval);
    command result_t stop();
    event result_t fired();
}
```

### ADC.nc

```cpp
interface ADC {
    async command result_t getdata();
    async command result_t getContinuousData();
    event result_t dataReady(uint16_t data);
}
```
Example Application

- A simple TinyOS application which periodically reads in the light intensity value, computes a moving average, displays it on the LED
SenseTaskM.nc

module SenseTaskM {
    provides {
        interface StdControl;
    }
    uses {
        interface Timer;
        interface ADC;
        interface StdControl as ADCControl;
        interface Leds;
    }
}

Diagram:

- SenseTask
  - start/stop to Timer
  - dataReady to ADC
  - fired to ADC
- Timer
  - start/stop
- ADC
  - getData
- Leds
  - xxxOn/Off
- ADCControl
  - init/start/stop
Module: Implementation

- Define
  - commands and event handlers
  - frame (storage)
    - statically allocated, fixed size to know memory requirement and avoid overhead of dynamic allocation

- See SenseTaskM.nc
Explicit Linking of Components

Two types of components:

- **modules**: individual components
- **configurations**: assemble components together, connecting interfaces (objects) used by components to interfaces (objects) provided by others
  - See SenseTask.nc
TinyOS Execution Model

SenseTask

Timer

ADC

ADCControl

LED

start/stop

fired

dataReady

xxxOn/Off()

init/start/stop

Design options?
TinyOS Execution Model

- Concurrency model: only two threads
  - long running tasks that can be interrupted by hardware event handlers

- Tasks perform the primary computation work
  - commands and event handlers post tasks
  - call lower level commands
  - signal higher level events
  - schedule other tasks within a component

- Tasks are posted to a FIFO queue
  - each task is atomic with respect to other tasks
  - run to completion, but can be preempted by events
  - the task scheduler is a simple FIFO scheduler

See SenseTaskM.nc details
Running tinyOS Program

- make mica
- ncc -o main.exe -target=mica SenseTask.nc
- avr-objcopy --output-target=srec main.exe main.srec
- Use uisp to install
A More Complete Sample Application

- Sensor network monitoring
  - monitor temperature and light conditions
  - periodically transmit measurements to a base station
  - sensors can forward data for other sensors that are out of range of the base station
  - dynamically determine the correct routing topology for the network
Internal Component Graph

application

Ad hoc Routing Application

Active Messages

Radio Packet

UART Packet

Temp

I2C

Light

Clocks

RFM

UART

bit

packet

byte
Message Send Transition

- Total propagation delay up the 5 layer radio communication stack is about 80 instructions

Timing diagram of event propagation
Evaluation: Storage

- Scheduler only occupies 178 bytes

- Complete application only requires 3 KB of instruction memory and 226 bytes of data (less than 50% of the 512 bytes available)

- Only processor_init, TinyOS scheduler, and C runtime are required

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Code Size (bytes)</th>
<th>Data Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Routing</td>
<td>88</td>
<td>0</td>
</tr>
<tr>
<td>AM_dispatch</td>
<td>40</td>
<td>0</td>
</tr>
<tr>
<td>AM_temperature</td>
<td>78</td>
<td>32</td>
</tr>
<tr>
<td>AM_light</td>
<td>146</td>
<td>8</td>
</tr>
<tr>
<td>AM</td>
<td>356</td>
<td>40</td>
</tr>
<tr>
<td>RADIO_packet</td>
<td>334</td>
<td>40</td>
</tr>
<tr>
<td>RADIO_byte</td>
<td>810</td>
<td>8</td>
</tr>
<tr>
<td>RFM</td>
<td>310</td>
<td>1</td>
</tr>
<tr>
<td>Light</td>
<td>84</td>
<td>1</td>
</tr>
<tr>
<td>Temp</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>UART</td>
<td>196</td>
<td>1</td>
</tr>
<tr>
<td>UART_packet</td>
<td>314</td>
<td>40</td>
</tr>
<tr>
<td>I2C</td>
<td>198</td>
<td>8</td>
</tr>
<tr>
<td>Processor_init</td>
<td>172</td>
<td>30</td>
</tr>
<tr>
<td>TinyOS scheduler</td>
<td>178</td>
<td>16</td>
</tr>
<tr>
<td>C runtime</td>
<td>82</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>3450</td>
<td>226</td>
</tr>
</tbody>
</table>
## Evaluation: Timing

<table>
<thead>
<tr>
<th>Operations</th>
<th>Cost (cycles)</th>
<th>Time (µs)</th>
<th>Normalized to byte copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte copy</td>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Post an Event</td>
<td>10</td>
<td>2.5</td>
<td>1.25</td>
</tr>
<tr>
<td>Call a Command</td>
<td>10</td>
<td>2.5</td>
<td>1.25</td>
</tr>
<tr>
<td></td>
<td>46</td>
<td>11.5</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>51</td>
<td>12.75</td>
<td>6</td>
</tr>
<tr>
<td>Post a task to scheduler</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Context switch overhead</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interrupt (hardware cost)</td>
<td>9</td>
<td>2.25</td>
<td>1</td>
</tr>
<tr>
<td>Interrupt (software cost)</td>
<td>71</td>
<td>17.75</td>
<td>9</td>
</tr>
</tbody>
</table>
Summary: TinyOS

- **Components**
  - provide commands and require callback hooks for event-driven programming

- **Configurations**
  - Link components

- **TinyOS**
  - an app (configuration) at a time, linking only necessary components

- **Two threads exec**
  - one for event
  - one for task

```c
interface ADC {
    async command result_t getdata();
    async command result_t getContinuousData();
    event result_t dataReady(uint 16_t data);
}
```

```c
configuration SenseTask {
    // this module does not provide any interfaces
}
implementation {
    components Main, SenseTaskM, LedsC, TimerC, DemoSensorC as Sensor;

    Main.StdControl -> TimerC;
    Main.StdControl -> Sensor;
    Main.StdControl -> SenseTaskM;

    SenseTaskM.Timer ->
        TimerC.Timer[unique("Timer")];
    SenseTaskM.ADC -> Sensor;
    SenseTaskM.Leds -> LedsC;
}
```
Discussion: Compare GNURadio/Sora/TinyOS

- What are some similar software concepts?
- What are some differences?
Discussion

- Can we use GNURadio/SORA/TinyOS for writing mobile applications for mobile phones/tablets, or in other words, what are missing?
Outline

- Admin and recap
- Wireless and mobile software framework
  - GNURadio
  - SORA
  - TinyOS
  - Java ME
Java Platforms

Java is divided into multiple platforms, e.g.,

- Enterprise Edition (EE)
  - business applications

- Standard Edition (SE)
  - general applications

- Micro Edition (ME)
  - small devices such as mobile phone, PDA, car navigation

Oracle’s claims on Java on mobile devices
To accommodate heterogeneous mobile devices, define **configurations and profiles**

- A *configuration* provides fundamental services for a broad category of devices (e.g., lang, io, util)
- A *profile* supports higher-level services common to a more specific class of devices or market (e.g., life cycle, GUI)
- An *optional package* adds specialized services that are useful on devices of many kinds, but not necessary on all of them

http://www.oracle.com/technetwork/java/embedded/javame/overview/index-jsp-138655.html
Example ME Configurations

- Connected Limited Device Configuration (CLDC)
  - 160 KB to 512 KB of total memory available
  - 16-bit or 32-bit processor
  - low power consumption and often operating with battery power
  - connectivity with limited bandwidth
  - examples: cell phones, certain PDAs

- Connected Device Configuration (CDC)
  - 2 MB or more memory for Java platform
  - 32-bit processor
  - high bandwidth network connection, most often using TCP/IP
  - examples: set-top boxes, certain PDAs
CLDC Classes

- Boolean
- Byte
- Character
- Class
- Integer
- Long
- Math
- Object
- Runnable
- Runtime
- Short
- String
- StringBuffer
- System
- Thread
- Throwable

java.lang

java.util

java.io
Example ME Profiles

- **Mobile Information Device Profile (MIDP)**
  - GUI, multimedia and game functionality, end-to-end security, and greater networked connectivity
  - mobile phones and entry level PDAs with display

- **Foundation Profile**
  - set of Java APIs that support resource-constrained devices without a standards-based GUI system

- **Personal Profile**
  - Full set of AWT APIs, including support for applets and Xlets
  - CDC + Foundation Profile + Personal Profile for high-end PDA

...
MIDP Packages

- java.io
- java.lang
- java.util
- javax.microedition.io
- javax.microedition.lcdui
- javax.microedition.rms
- javax.microedition.midlet
- javax.microedition.lcdui.game
- javax.microedition.media
- javax.microedition.media.control
- javax.microedition.pki

version 1.0

addition in version 2.0
MIDP Technology Stack

- **Your MIDlet**
  - Yellow Pages, train schedules and ticketing, games...
  - UI, HTTP networking...

- **Mobile Information Device Profile**

- **J2ME core APIs**

- **KVM**
  - Threads, no Floats...
  - 32-bit RISC, 256K ROM, 256K Flash, 64K RAM

- **DSP chip (e.g., ARM)**

- **CLDC = KVM + J2ME Core APIs in this example**
What are some major differences between desktop GUI app design and mobile (phone/tablet) GUI app design?
Some Key Features of Mobile UI App

- Limited screen real estate
  - one thing at a time

- Limited resources
  - need more dynamic system management on app life cycle; give app chances to adapt, better mem management
MIDlet

- An MIDP application is called a MIDlet
  - similar to the SE applet

- A MIDlet moves from state to state in the lifecycle, as indicated
  - start - acquire resources and start executing
  - pause - release resources and become quiescent (wait)
  - destroy - release all resources, destroy threads, and end all activity
MIDP Visual Display

- Each MIDP has one instance of Display
  - `Display.getDisplay(this)` to get the manager
  - At any instance of time at most one Displayable object can be shown on the display device and interact with user
    - `display.setCurrent(<Displayable object>)`
MIDP Visual Display

- Displayable
  - Canvas
    - GameCanvas
  - Screen
    - Alert, List, TextBox, Form

- Form can contain multiple form items for organization
  - Labels, Image Items, String Items, Text Fields, Date Fields, Gauges, Choice Groups
MIDP: User Interaction

- Displayable objects can declare commands and declare a command listener:
  - `addCommand(Command cmd)`
  - `addCommandListener()`

- `Command(<label>, <type>, <priority>)`
  - **Type**: BACK, CANCEL, EXIT, HELP, ITEM, OK, SCREEN, and STOP
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class HelloWorldMIDlet extends MIDlet implements CommandListener {
    private Command exitCommand;
    private Display display;
    private TextBox t;

    public HelloWorldMIDlet() {
        display = Display.getDisplay(this);
        exitCommand = new Command("Exit", Command.EXIT, 2);
        t = new TextBox("CS434", "Hello World!", 256, 0);
        t.addCommand(exitCommand);
        t.setCommandListener(this);
    }

    public void startApp() { display.setCurrent(t); }
    public void pauseApp() { }
    public void destroyApp(boolean unconditional) { }
    public void commandAction(Command c, Displayable s) {
        if (c == exitCommand) {
            destroyApp(false);
            notifyDestroyed();
        }
    }
}
Summary: Java ME MIDP Architecture

Lifecycle callbacks
- startApp
- pauseApp
- destroyApp

disp=Display.getDisplay(this)
disp.setCurrent(disp)

disp.addCommand()
MIDP: Persistent State

- Record store defined in javax.microedition.rms

- Record store identified by name:
  - static String[] listRecordStores();

  - recordStore = RecordStore.openRecordStore("scores", true);

  - recordId = addRecord(byte[] data, int offset, int numBytes);

  - getRecord(int recordId);
Summary: Java ME

- Scale down a popular programming environment to ease learning
- Use virtual machines to mask device heterogeneity
- Use configuration/profiling to handle device heterogeneity and avoid using lowest common denominator
- MIDLet to manage app life cycle
- Displayable to visual display, commands and provides command listener
- Introduce persistent record store
Discussion on Java ME

- What designs of Java ME do you like and hence expect later frameworks (e.g., iOS, Android) may have too?

- What features do you think are missing in Java ME?
Common Features We Expect of Mobile UI App Framework

- Limited screen real estate
  - one thing at a time

- Limited resources: more dynamic system management on app life cycle
  - give app chances to adapt, better mem management
Mobile GUI App

- Screen real-estate is limited => Focus on one thing at a time
Mobile GUI App

- Screen real-estate is limited => Focus on one thing at a time
Mobile GUI App Workflow

App lifecycle callbacks/custom
- start
- pause
-....
Mobile GUI App Workflow: Do One Thing

App lifecycle callbacks/custom
- start
- pause
- ...

Diagram:
- App
  - Display Composite
    - Display
    - Display Composite
      - Display
      - Display
    - Display
Mobile GUI App Workflow: Switch to Another GUI

App lifecycle callbacks/custom
- start
- pause
- ...

App

Display Composite

Display

Display Composite

Display

Display Composite

Display

Display

Display Composite

Display

Display Composite

Display

Display
Mobile GUI App Workflow: Display Content Based on Underlining Data

App lifecycle callbacks/custom
- start
- pause
- ...

[Diagram showing app lifecycle and display content based on underlining data]
Mobile GUI App Workflow: Handle Events

App lifecycle callbacks/custom
- start
- pause
- ...

App

Display Composite

Display

Display Composite

Display

Display

Display Composite

Display

Display

Data/Model

Event Handler

Event Handler

Data/Model