Dandelion

Transparently programming phone-centered body sensor applications

Felix XiaoZhu Lin, Ahmad Rahmati, and Lin Zhong
Rice Efficient Computing Group
**Goal**: transparently develop body sensor app

**Challenge**: the difficulty in programming sensors

**Impact**: ecosystem with numerous phone developers
Popular phone applications

225,000

72,000

Source: Mobile Developer Economics 2010 and Beyond. Produced by VisionMobile. Sponsored by Telefonica Developer Communities. June 2010. Licensed under Creative Commons Attribution 3.0 License. Any use or remix of this work must retain this notice.
6000+ for health

Phone + body sensors

= Innovative apps!
How many apps utilize body sensors?

~%0
Why?

Body sensors **different** and **difficult** to program
### Phone
- 32-bit ARM processor
- Java/C++/ObjC
- Linux/iOS

### Sensor
- 16-bit microcontroller
- Procedural, domain-specific
- TinyOS/uCOS
```c
uint16_t avg_energy = 0;
OS_EVENT *event_queue, *ack_event;

void init(void){
    /* Init event queue, timer, etc (~XXX lines) */
}

/* sent an alarm packet to main body */
void send_alert_packet(void) {
    do {
        /* Create a FALL_ALARM packet, send to the main
         body, and wait for ACK (-10 lines) */
        send_packet(/* pkt */);
        OSSemPend(ack_event, timeout, &err);
    } while (err);
}

void on_rx_packet (void) {
    /* Recv and decode incoming packet (~10 lines)*/
    if (/* a new command from the smartphone */)
        OSSemPost(event_queue, cmd_event);
    else /* ACK for a previous alert packet */
        OSSemPost(ack_event);
}

void on_timer_expire (void) {
    /* Create a timer event and post to the main
     event loop (~10 lines) */
    OSSemPost(event_queue, event);
}

void main(void) {
    uint16_t energy;
    event_t * event = 0;
    while(1) {
        /* blocking wait for new event */
        event = OSSemPend(event_queue, timeout, &err);
        switch (event->type) {
        case EV_TIMER: /* timer-driven sampling */
            energy = sensors(0)*sensors(0) +
                     sensors(1)*sensors(1) +
                     sensors(2)*sensors(2);
            avg_energy = avg_energy / 2 + energy;
            if (avg_energy > THRESHOLD) {
                send_alert_packet();
            }
            break;
        case EV_UART0_CMD: /* command from main body */
            packet_t *pkt = (packet_t *)event;
            /* decode the packet as command ... */
            stop_timerA();
            break;
        }
    }
}
```
Phone

Sensor

Ease

Programming style

Dumb supplier

Efficiency

In-phone

In-sensor

Processing
Sensor as *dumb* data supplier

- Phone developers
- Sensor vendor

Diagram:

- Application
- Phone
- Sensor
- Raw data
Phone

Sensor

Ease

Very energy-inefficient

Dumb supplier

Programming style

Processing

In-phone

In-sensor

Efficiency
In-sensor

In-phone

Processing

Efficiency

Ease

Sensor

Phone

Dumb supplier

Directly program
Directly program the sensor

Phone developers

Application

Phone

Proc’d results

Sensor code

Sensor
Phone

Sensor

Ease

Programming style

Dumb supplier

Not developer-friendly

Directly program

In-phone

In-sensor

Processing

Efficiency
Fixed features

Phone developers

Sensor vendor

Application

Phone

Feature library

Sensor

Features
Ease

Phone

Sensor

Programming style

In-phone

In-sensor

Processing

Efficiency

Dumb supplier

Fixed features

Directly program

No full programmability
Dandelion

Transparently develop body sensor apps
### Phone
- 32-bit ARM processor
- Java/C++/ObjC
- Transparency
- Linux/iOS

### Sensor
- 16-bit microcontroller
- C
- TinyOS/uCOS
- iOS
Phone developers

Phone app

App body

Phone

Transparent Integration

Senselet

runtime

Sensor

Sensor vendor
System Design

The runtime system
Programming abstraction
Programming support
The runtime system

Phone

Wireless

Senselet

Runtime

Sensor

Senselet

Runtime

Sensor

App body

Runtime
Senselet

Programming abstraction

SenseletBase

Template

MySenselet1
- Methods
- Private states

Data processing

MySenselet2
- Methods
- Private states

Data processing
Programming Support

Platform services
Remote method invocation
Compile & deploy
Three platform services

Essential + widely supported

Data acquisition
Timer
Dynamic memory
Remote method

Function calls across a senselet and the app body

```c
int data[128];
...
PutResults(data, 128);
...

void PutResults (int *data, int len) {
  // … deal with the data
}
```
Target-independent compilation

IR: Intermediate Representation
class SenseletFall : public SenseletBase {
    public:
    void OnCreate() {_avg_energy = 0; RegisterSensorData(ACCEL, 36);};

    void OnData (uint8_t *readings, uint16_t len) {
        uint16_t energy = readings[0]*readings[0] + readings[1]*readings[1] + 
                        readings[2]*readings[2];
        //do a simple low-pass filtering
        _avg_energy = _avg_energy / 2 + energy / 2;

        // detect fall accident with the filtered energy
        if (_avg_energy > THRESHOLD) { theMainBody.FallAlert(); }
    }

    void OnDestroy () {UnRegisterSensorData(ACCEL);}

private:
    uint16_t _avg_energy;
};
Implementation

Platform + Dandelion + body sensor apps

Nokia N900
Rice Orbit body sensor
LLVM compiler infrastructure
Source code comparison
Three apps, each with three implementations

# of source lines

Portable, concise code

FallDetector  EKG  Pedometer

Dandelion  Embedded OS  Bare-bone
Memory overhead
Measured on MSP430
Execution overhead

Measured on MSP430

Processor cycle per period

Dandelion Processing

<3

IDLE
Conclusions

• Dandelion enables transparently programming body sensor apps

• Dandelion incurs very small overhead
The Ecosystem

- Developers
- Vendors
- Users
http://www.cs.rice.edu/~xl6/dandelion/

THANKS!