JAVA™ ON WIRELESS SENSORS

Cristina Cifuentes
Sun Labs

February 2006
Agenda

• Wireless Sensor Networks
• Proposed Solution and Demo
• The Squawk Java™ Virtual Machine
• The Sun™ Small Programmable Object Technology (SPOT) System
• The Wireless API
• Conclusions and Future
The State of the Art

• Ideas of “Smart Dust”
• Berkeley motes, TinyOS, IEEE 802.15.4
• Sun Labs Anteater project
  > Research into impact and meaning of such systems to Sun Microsystems
  > Major customer advantage seen as economics and flexibility
• Most of the work is aimed at infrastructure issues
  > Size, power, and networking (mesh networking)
Applications: Chicken and Egg

• Hard to develop applications using current technologies
  > Low-level C-like languages
  > Unproductive development tools
    • Hardly any debugging support
  > Too many low-level concerns in current systems
    • Most high-level software developers do not know how hardware works, or even have an appreciation any more
  > Not accessible to majority of software developers
Agenda

• Wireless Sensor Networks
• Proposed Solution and Demo
• The Squawk Java™ Virtual Machine
• The Sun™ Small Programmable Object Technology (SPOT) System
• The Wireless API
• Conclusions and Future
Proposed Solution

Squawk Java VM

CLDC libraries

Libraries for driving hardware: radio, sensor board

Network libraries: 802.15.4, GCF implementation

Desktop libraries: connect from J2SE VMs to wireless devices

Hardware: 32-bit ARM core, Chipcon CC2420 based wireless platform, SPI based peripherals

From conception to implementation: 6 months
public static void main(String[] args) throws IOException {
    // Setup and read from accelerometer
    Accelerometer3D acc = DemoSensorBoard.getAccelerometer();
    ((LIS3L02AQAccelerometer) acc).set6GScale();
    RangeInput x = acc.getX(), y = acc.getY(), z = acc.getZ();

    SensorBoardColouredLED led = SensorBoardColouredLED.getLed1();
    led.setOn(); // switch it on
    led.setRGB(0,0,0); // ... but black it out

    // Display red/green/blue on LED based on motion difference
    int lastX = 0, lastY = 0, lastZ = 0;
    while(true) {
        int xValue = x.getValue(), yValue = y.getValue(), zValue = z.getValue();
        int r = Math.abs(xValue-lastX) > 10 ? 255:0;
        int g = Math.abs(yValue-lastY) > 10 ? 255:0;
        int b = Math.abs(zValue-lastZ) > 10 ? 255:0;
        led.setRGB(r,g,b);

        lastX = xValue; lastY = yValue; lastZ = zValue;
    }
}
Agenda

• Wireless Sensor Networks
• Proposed Solution and Demo
• The Squawk Java™ Virtual Machine
• The Sun™ Small Programmable Object Technology (SPOT) System
• The Wireless API
• Conclusions and Future
Standard JVM vs Squawk JVM

Standard JVM
- Java class library
- Loader
- Verifier
- Garbage collector
- Interpreter
- Thread Scheduler
- Compiler
- I/O library
- Native code

Squawk JVM
- Java class library
- Loader
- Verifier
- Transformer
- Garbage collector
- Interpreter
- Thread Scheduler
- Exporter
- Compiler
- Device Driver Architecture
- I/O library
- Native code

Diagram showing the differences between Standard JVM and Squawk JVM in terms of components and code types.
The Squawk JVM

- Java VM mainly written in Java
  - Interpreter written in C
  - Garbage collector translated from Java to C
- J2ME level VM
  - CLDC 1.0/1.1 libraries
- Extra features
  - Runs on the bare ARM without an underlying OS
  - Interrupts and device drivers written in Java
  - Support application migration (extension to isolates),
- Memory footprint on the ARM:
  - 80K RAM for VM
  - 270K flash for libraries
Squawk Features for Wireless Sensor Devices

- Designed for memory constrained devices
- Runs on the bare metal on the ARM
- Represents applications as objects
- Runs multiple applications in the one VM
- Migrates applications from one device to another
- Authenticates deployed applications on device
Squawk’s Split VM Architecture

**.class**
- Loader
- Verifier
- Transformer (Optimizer)

Squawk bytecodes
- Exporter

**.suite**
- Thread Scheduler
- Garbage Collector
- Interpreter
- Compiler
- Device Driver Architecture
- Java class libraries
- I/O library
- native code

machine code
### Squawk Bytecodes: Designed for Memory Constrained Devices

<table>
<thead>
<tr>
<th>Squawk Bytecode Property</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commonly used bytecodes are 2 bytes instead of 3 bytes</td>
<td>More compact</td>
</tr>
<tr>
<td>References to fields and methods resolve into physical offsets</td>
<td>More efficient for interpretation</td>
</tr>
<tr>
<td>Local variables are typed</td>
<td>More efficient for compilation</td>
</tr>
<tr>
<td>One OOP map per method, nothing on the operand stack at GC points</td>
<td>Simplifies garbage collection</td>
</tr>
<tr>
<td></td>
<td>Eliminates need for static interpretation to decipher activation frames</td>
</tr>
</tbody>
</table>
Suite Files

• Preprocessed set of classfiles
• Internally fully linked
  > Pointers to other classes in suite or parent(s) only
  > Chain of suites is a transitive class closure
• Uses Squawk bytecode set
# Classfiles vs Suite Files Size Comparison

<table>
<thead>
<tr>
<th>Application</th>
<th>JAR</th>
<th>Suite</th>
<th>Suite/JAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLDC</td>
<td>458,291</td>
<td>149,542</td>
<td>0.33</td>
</tr>
<tr>
<td>cubes</td>
<td>38,904</td>
<td>16,687</td>
<td>0.42</td>
</tr>
<tr>
<td>hanoi</td>
<td>1,805</td>
<td>835</td>
<td>0.46</td>
</tr>
<tr>
<td>delta blue</td>
<td>30,623</td>
<td>8,144</td>
<td>0.27</td>
</tr>
<tr>
<td>mpeg</td>
<td>100,917</td>
<td>54,888</td>
<td>0.54</td>
</tr>
<tr>
<td>manyballs</td>
<td>12,017</td>
<td>6,100</td>
<td>0.51</td>
</tr>
<tr>
<td>pong</td>
<td>17,993</td>
<td>7,567</td>
<td>0.42</td>
</tr>
<tr>
<td>spaceinvaders</td>
<td>50,854</td>
<td>25,953</td>
<td>0.51</td>
</tr>
<tr>
<td>tilepuzzle</td>
<td>18,516</td>
<td>7,438</td>
<td>0.40</td>
</tr>
<tr>
<td>wormgame</td>
<td>23,985</td>
<td>9,131</td>
<td>0.38</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>753,905</strong></td>
<td><strong>286,285</strong></td>
<td><strong>0.38</strong></td>
</tr>
</tbody>
</table>
Squawk Features for Wireless Sensor Devices

- Designed for memory constrained devices
- Runs on the bare metal on the ARM
- Represents applications as objects
- Runs multiple applications in the one VM
- Migrates applications from one device to another
- Authenticates deployed applications on device
Interrupt Handling and Device Driver Support

- Device driver sets up the interrupt controller
- Interrupt handler thread blocks waiting for the VM to signal an event
- When an interrupt occurs, an assembler interrupt handler sets a bit in an interrupt status word and disables the interrupt
- At each VM reschedule point, the bit is detected, the event signaled, the scheduler resumes the interrupt handler thread, which handles the interrupt and re-enables it
- Device driver written in Java
Interrupt Latency

> Dependent on the time from the global interrupt handler running until the next VM schedule

> Optimal case
  • VM is idle => no penalty

> Average case
  • VM is executing bytecodes in another thread => VM reschedules after a certain number of back branches

> Worst case
  • VM is executing a GC => VM reschedules after the GC completes; in practice, < 1 msec
Squawk Features for Wireless Sensor Devices

- Designed for memory constrained devices
- Runs on the bare metal on the ARM
- Represents applications as objects
- Runs multiple applications in the one VM
- Migrates applications from one device to another
- Authenticates deployed applications on device
OS Processes and Isolates Analogy

**Operating System**

Process

Resources

- thread1
- thread2
- thread3

Process

Resources

- thread1
- thread2
- thread3

**Squawk JVM**

Isolate

Resources

- thread1
- thread2
- thread3

Isolate

Resources

- thread1
- thread2
- thread3
Squawk Isolates

• Each application is represented by an Isolate object
• Similar to JSR 121 Isolate API
  > Each isolate has resources that are shared amongst the threads of that isolate
  > Immutable state (e.g., methods, string constants, parts of classes) is shared
    • Non-shared class state includes static fields, class initialization state, and class monitors
  > Different support for inter-isolate communication
    • Uses generic connection framework
• Allows for reification of applications
  > Can start(), pause(), resume(), and exit()
public void run() throws Exception {
    ...
    Isolate isolate = new Isolate("com.sun.spots.SelfHibernator", url());
    isolate.start();
    send(isolate, outStream);
    ...
}
Squawk Features for Wireless Sensor Devices

- Designed for memory constrained devices
- Runs on the bare metal on the ARM
- Represents applications as objects
- Runs multiple applications in the one VM
- Migrates applications from one device to another
- Authenticates deployed applications on device
Multiple Isolates (Applications) on the One JVM

**Standard JVM**
- JVM OS process
- JVM OS process
- Non-shareable object memory
- Shareable object memory

**Squawk JVM**
- JVM OS process
- Isolate
- Isolate
- Non-shareable object memory
- Shareable object memory
Squawk Features for Wireless Sensor Devices

- Designed for memory constrained devices
- Runs on the bare metal on the ARM
- Represents applications as objects
- Runs multiple applications in the one VM
- Migrates applications from one device to another
- Authenticates deployed applications on device
Isolate (Application) Migration

• Isolates can be migrated
  > Migrates state of a running application, to continue running on a target device
  > Target device must have binary code (suite) of the application
  > Migration uses same object serialization mechanism as the suite creator
  > Constraints on external state
    • Must be none, or
    • Must be homogeneous at both ends (Sun SPOT Squawk), or
    • Must be serializable (desktop Squawk)
Squawk Features for Wireless Sensor Devices

- Designed for memory constrained devices
- Runs on the bare metal on the ARM
- Represents applications as objects
- Runs multiple applications in the one VM
- Migrates applications from one device to another
- Authenticates deployed applications on device
Secure Suite Deployment

• Digitally sign suites on the desktop and verify signature on the Sun SPOT at installation time

• Why?
  > Ensure that split VM architecture does not compromise verified Java applications on the desktop
    • Suite originated at a trusted source
    • Suite was received in the state intended

• For a user, each device is bound to one or more Sun SPOT SDK installations
Secure Suite Deployment

• Each SDK has an associated public/private key pair
  > Key pair automatically generated on the background first time the SDK is installed

• Private key used for signing
  > Stored on desktop in password protected file

• Public key stored on device

• Signed suites can also be migrated from device to device
Agenda

• Wireless Sensor Networks
• Proposed Solution and Demo
• The Squawk Java™ Virtual Machine
• The Sun™ Small Programmable Object Technology (SPOT) System
• The Wireless API
• Results
Sun SPOT Hardware

- ARM7 core
  - 256K RAM/2M ROM
- CC2420 radio
  - Strip antenna
- Single LED
- Double sided connector for stackable boards
- Can be powered from single 1.5V battery
  - Requires 25-90mA depending on operation
- 35x25 mm in size
- USB and OTA connection to desktop
Sun SPOT Demo Sensor Board

- Demo sensor board
  - 3D accelerometer
  - 9 I/O Pins (PWM capable)
  - Temperature sensor
  - Light sensor
  - IRDA serial connection

- All SPI driven peripherals

- Users can build own transducer boards
  - Experimental board available
Sun SPOT Software Libraries

• Standard J2ME Java libraries
  > CLDC 1.0/1.1

• Hardware libraries
  > SPI, AIC, TC, PIO drivers all in Java
  > Sensor board hardware driven by Java (no C)
    • ADCs, GPIO, IRDA, etc.

• Radio libraries
  > Drive Chipcon CC2420 hardware from Java (no C)
Software Libraries (2)

- Network libraries
  > 802.15.4 MAC layer in Java (no C)
  > Simple GCF implementations of connections

- Desktop libraries
  > Create connections from standard J2SE VMs to wireless devices
  > Utilize one Sun SPOT as a gateway (base station)
Sun SPOT Graphical UI: SpotWorld

NetBeans IDE 4.1 - MetaIsolate and More 06-14

Projects
- Base Station 06-14
- MetaIsolate and More 06-14
- reactomatic
- SpotWorld 06-14

Navigator - Blip

Members View
- exec()
- act()
- nil-SensorBoard()
- lightValue(int n)
- man(String args)
- sleep(int ms)
- b int
- g int
- led1 SensorBoardColouredLED
- led2 SensorBoardColouredLED
- t int

Filters:  

Output - SpotWorld (jar)

init:
deps-jar:

Building jar: C:\Sun\SunSpot\SpotWorld\dist\SpotWorld.jar
jar:
BUILD SUCCESSFUL (total time: 3 seconds)

Finished building SpotWorld (jar).

Created Isolate...
View Isolate Source
List Source Files
Ping
Disconnect
Reconnect
Bind
Verbosity on
Verbosity off

sspot://compomatic:26564
sspot://spotomatic:20564
Agenda

- Wireless Sensor Networks
- Proposed Solution and Demo
- The Squawk Java™ Virtual Machine
- The Sun™ Small Programmable Object Technology (SPOT) System
- The Wireless API
- Conclusions and Future
The Wireless API Building Blocks

- **IEEE 802.15.4**
  - Low data rates (250 kbps, 40 kbps, and 20 kbps)
  - Multi-month to multi-year battery life
  - Low complexity

- **Generic connection framework**
  - Part of J2ME
  - Hierarchy of interfaces and classes that create connections (e.g., HTTP, datagram) and perform I/O
  - \texttt{javax.microedition.io} package
The Wireless API - Radio

radio://{address}:{port}

- Streaming radio connection
- **address**: unique IEEE address of the device
- **port**: channel to be used
Radio Example

- Output number 5 to device 1020 on channel 42

```java
StreamConnection conn = (StreamConnection) Connector.open("radio://" + 1020 + ":42");
DataOutputStream output = conn.openDataOutputStream();
output.writeInt(5);
output.flush();
```
The Wireless API – Radiogram

radiogram:://{address} | broadcast:{port}

- Datagram-style communication
- Point-to-point
- Can also broadcast to multiple listeners
Radiogram Example

- Send radiogram to device 1020 on channel 42 and wait for receiving a datagram from remote device

```java
StreamConnection conn = (StreamConnection) Connector.open("radiogram://" + 1020 + ":42");
Datagram dg = conn.newDatagram(conn.getMaximumLength());
dg.writeUTF("Hello world");
conn.send(dg);
conn.receive(dg);
```
Radiogram Broadcast Example

- Listen for radiograms on port 51 from all neighbours and broadcast the received signal strength indicator (RSSI) to any listeners by broadcasting on port 52

```java
public NeighbourhoodSender() throws IOException {
    listenerConn = (DatagramConnection)Connector.open("radiogram://:51");
    inputPacket = (Radiogram)listenerConn.newDatagram(0);
    senderConn = (DatagramConnection)Connector.open("radiogram://broadcast:52");
    outputPacket = senderConn.newDatagram(senderConn.getMaximumLength());
}
```
Conclusions

- Java on “wireless sensor networks” is here
  - Small Java-based VM
    - Java runs on the bare metal, no underlying OS needed
  - Better developer experience than the state-of-the-art
    - Standard Java development and debugging tools
    - Simple out-of-the-box experience (SpotWorld)
  - Mid-level sensor device that can be battery powered
    - Enable exploratory programming
    - Enable more on device computation and reduce network traffic
    - Enable over-the-air programming
Future

• Collaborate with qualifying partners
• Use within Sun Labs
  > Gesture based interfaces, building instrumentation, self-organising systems, etc.
• Iterate hardware design
  > Smaller chips, lower power, cheaper, etc.
• Iterate VM
  > Smaller footprint, faster, smarter interrupts, power management, etc.
• Productize
• Open schematics and VM to the community?
The Teams

• Squawk
  > Nik Shaylor (alumni)
  > Bill Bush (alumni)
  > Doug Simon (alumni)
  > Cristina Cifuentes
  > Derek White
  > Eric Arseneau

• Squawk ARM Support
  > John Daniels
  > Dave Cleal
  > Duncan Pierce
  > Rachel Davies
  > John Wilcox

• Sun SPOT Hardware
  > Bob Alkire
  > John Nolan (alumni)
  > Del Peck

• Sun SPOT Software
  > Randy Smith
  > Bernard Horan (alumni)
  > Vipul Gupta
  > Samita Chakrabarti
  > Rob Tow
  > David Simmons

• Managers
  > Roger Meike (Sun SPOT)
  > Dan Ingalls (Squawk)
Questions