Tools for Distributed Embedded Systems

Jan Beutel, ETH Zurich
National Competence Center in Research – Mobile Information and Communication Systems
Preliminaries - Goals of this Lecture

• Talk about system building for Wireless Sensor Networks

• Understand what is actually hard about WSN applications

• Show what can (and cannot) be done with today’s tools, support and methodology

• Mix of “standard” tools and methodology, adapted, abridged and some of our (own) “new” research.

• Although there is an ordering, many topics discussed can be used standalone, depending on the problem to be solved.
Outline

• Wireless Sensor Networks – Platforms and Architecture

• Building Applications – PermaSense Example

• Differentiating Factors

• Tools for Distributed Embedded Systems
  - Compilation & Analysis
  - Execution on Testbeds
  - Passive Inspection
  - Physical Emulation
  - Automated Validation

• Concluding Experience – Examples
Wireless Sensor Networks

PLATFORMS AND ARCHITECTURE
Basic Concepts of Wireless Sensor Network Platforms

• “Mote class” devices
  - Microcontroller + low-power radio
  - Battery powered
  - Many custom applications
  - Large design space, many variants
  - Most prominent examples: Mica2, Mica2Dot, Tmote Sky

• Hardware is packaged with
  - System software and apps
  - Base stations, network access
  - Server-side solutions (backends)
  - Tools (e.g. simulators, …)
A Popular Software System - TinyOS

- Event driven “Operating System”
  - Basically just a collection of drivers and an event queuing system
  - Event handlers must not block...
  - Written in nesC, a C dialect
  - Compositional nature (hw/sw components, interfaces)
  - Geared towards simple applications (max. 10k RAM, 40 k ROM)

- De-facto standard in WSN applications
  - Popular in academia and industry

- Many (comparable) systems: Contiki, SOS, Mantis, BTnut...
Building Real WSN Applications

THE PERMASENSE PROJECT IN A NUTSHELL
PermaSense - Alpine Permafrost Monitoring

- Cooperation with Uni Basel, Uni Zurich and EPFL
PermaSense - Aims and Vision

Geo-science and engineering collaboration aiming to:
- provide long-term high-quality sensing in harsh environments
- facilitate near-complete data recovery and near real-time delivery
- obtain better quality data, more effectively
- obtain measurements that have previously been impossible
- provide relevant information for research or decision making, natural hazard early-warning systems
Understanding When and Why Catastrophes Happen

Eiger east-face rockfall, July 2006, images courtesy of Arte Television
PermaSense Deployment Sites 3500 m a.s.l.

A **scientific instrument** for **precision sensing** and data recovery in environmental extremes
PermaSense - Key Architectural Requirements

• Support for ~25 nodes

• Different sensors
  – Temperatures, conductivity, crack motion, ice stress, water pressure
  – 1-60 min sensor duty-cycle

• Environmental extremes
  – −40 to +65°C, ΔT ≤ 5°C/min
  – Rockfall, snow, ice, rime, avalanches, lightning

• Near real-time data delivery

• Long-term reliability
  – ≥99% data yield
  – 3 years unattended lifetime

Relation to other WSN projects

• Comparable to other environmental monitoring projects
  – GDI [Szewczyk], Glacsweb [Martinez], Volcanoes [Welsh], SensorScope [Vetterli], Redwoods [Culler]

• Lower data rate
• Harsher, higher yield & lifetime
• Data quality/integrity
PermaSense - System Architecture

Sensor network

Base station

Backend
PermaSense - Base Station Overview

- Powerful embedded Linux (Gumstix)
- 4 GB storage
- Solar power (2x 90W, 100 Ah, ~3 weeks)
- WLAN/GPRS connectivity, backup modem
Base Station Sandwiched with WLAN Router

Gumstix Verdex
TinyNode
WLAN Router
GSM
EMP Protectors
IP68 Enclosure and Connectors
PermaSense - Sensor Node Hardware

- **Shockfish TinyNode584**
  - MSP430, 16-bit, 8MHz, 10k SRAM, 48k Flash
  - LP radio: XE1205 @ 868 MHz

- **Waterproof housing and connectors**

- **Protective shoe, easy install**

- **Sensor interface board**
  - Interfaces, power control
  - Stabilized measurements
  - 1 GB memory

- **3-year life-time**
  - Single battery, 13 Ah
  - ~300 µA power budget
Dozer Low-Power System Software Integration

- **Dozer ultra low-power data gathering system**
  - Beacon based, 1-hop synchronized TDMA
  - Optimized for ultra-low duty cycles
  - **0.167%** duty-cycle, **0.032mA**

- **System-level, round-robin scheduling**
  - “Application processing window” between data transfers and beacons
  - Custom DAQ/storage routine

[Burri – IPSN2007]
PermaDozer - Power Performance Analysis

<table>
<thead>
<tr>
<th>Operating Mode Characterization</th>
<th>[mA]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sleep</td>
<td>0.026</td>
</tr>
<tr>
<td>DAQ active(^a)</td>
<td>2.086</td>
</tr>
<tr>
<td>Dozer RX idle</td>
<td>13.64</td>
</tr>
<tr>
<td>Dozer RX</td>
<td>14.2</td>
</tr>
<tr>
<td>Dozer TX</td>
<td>54.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Measured Average Values</th>
<th>[mA]</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAQ only (2min)</td>
<td>0.110</td>
</tr>
<tr>
<td>Dozer only (30sec/2min)(^b)</td>
<td>0.072</td>
</tr>
<tr>
<td>PermaDozer total (30sec/2min)</td>
<td>0.148</td>
</tr>
</tbody>
</table>

\(^a\) Averages power consumption measured over a complete DAQ routine execution without attached sensor

\(^b\) Dozer only includes communication, not including network initialization and access to flash memory

148 uA average power
What Makes Building WSNs Different (so hard)?

DIFFERENTIATING FACTORS
Wireless Networked Embedded Systems

Unreliable Communication

Resource Constraints

Embedding in Environment

Distributed State

NCCR MICS
National Competence Center in Research
Mobile Information and Communication Systems
Sensornets Are Hard

- Sensor networks often fail/operate poorly
  - Great Duck Island network: median yield 58% [SenSys 2004]
  - Redwood network: median yield 40% [SenSys 2005]
  - Volcano network: median yield:68% [OSDI 2006]
- Survey of causes
  - Protocol conflicts/interference
  - Collisions and congestion induced loss
  - Neighbor management (with layer 2 scheduling, e.g. TMAC)
    Don't know!
- Low-power, limited resources make complete logging prohibitively expensive...
WSN Development Reality (ca. 2000)

- It is an art to deploy anywhere beyond 10-20 nodes.

- Need for concerted design, test and validation tools.
WSN Design and Development Peculiarities

• The microcontrollers typically used are ugly beasts
  – Very limited machines
  – Many special cases to consider (exceptions)

• Limited support through hardware
  – But many, often complex subsystems
  – Typically based on shared resources
  – Lack of abstractions

• Complex applications
  – Maybe uC are the wrong choice for certain apps…

• Influence of the environment (interaction)
WSN Design and Development Peculiarities Cont.

- Visibility of what is going on
- “Disconnected medium”
  - Getting information in and out of the network
- Low-power modes
- Lack of comprehensive analysis
  - Often there is lot’s of data but no clue what to do with it
- Lack of models to accompany/complement design effort
TOOLS FOR DISTRIBUTED EMBEDDED SYSTEMS
Preliminaries: Integrity of the Codebase

COMPILATION & ANALYSIS
Continuous Integration - Automated SW Builds

• Continuous Build and Test Loop
  – Watch repository for changes
  – Run compilation and tests
  – Publish status of builds and tests to developers

• Benefits
  – Detect errors early
  – Monitor code quality over time
  – Prevent integration problems
  – „Social“ factors
Regular Builds with Statistics

Number of Build Attempts: 296
Number of Broken Builds: 74
Number of Successful Builds: 222

Timeline of build types

Breakdown of build types

Timeline of coding violations
Email Notifications on Failures
Integration of Analysis Tools

FindBugs Report

Metrics

721 lines of code analysed, in 16 classes, in 1 packages.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Total</th>
<th>Density*</th>
</tr>
</thead>
<tbody>
<tr>
<td>High Priority Warnings</td>
<td>NaN</td>
<td></td>
</tr>
<tr>
<td>Medium Priority Warnings</td>
<td>5</td>
<td>6.90</td>
</tr>
<tr>
<td>Total Warnings</td>
<td>5</td>
<td>6.90</td>
</tr>
</tbody>
</table>

(* Defects per Thousand lines of non-commenting source statements)

Summary

<table>
<thead>
<tr>
<th>Warning Type</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bad practice Warnings</td>
<td>1</td>
</tr>
<tr>
<td>Performance Warnings</td>
<td>4</td>
</tr>
<tr>
<td>Total</td>
<td>5</td>
</tr>
</tbody>
</table>

Warnings

Click on a warning row to see full context information.

Bad practice Warnings

<table>
<thead>
<tr>
<th>Code</th>
<th>Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>So</td>
<td>Class Crash defines non-transient non-serializable instance field parent</td>
</tr>
</tbody>
</table>

Performance Warnings

<table>
<thead>
<tr>
<th>Code</th>
<th>Warning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bx</td>
<td>Method Window.setPos() invokes inefficient Integer(int) constructor, use Integer.valueOf(int) instead</td>
</tr>
<tr>
<td>Bx</td>
<td>Method Window.setVisible(int) invokes inefficient Integer(int) constructor, use Integer.valueOf(int) instead</td>
</tr>
<tr>
<td>SBCS</td>
<td>Method O Kills:OpeningFileString concatenates strings using a loop</td>
</tr>
<tr>
<td>Df</td>
<td>Unread field: node.id</td>
</tr>
</tbody>
</table>
Graphical Reporting of Results

TinyOS Memory Usage

Ram Usage

Flash Usage

Stack Usage
Regression Testing Using Continuous Integration

On code change applications are built from scratch and analyzed
- Standard practice in enterprise level software development
- Deeper understanding of long term development trends
- Service to the TinyOS community, increasing software quality

+5000 TinyOS-2.x regression builds over the last 2.5 years at ETHZ

[http://tik42x.ee.ethz.ch:8080]
Getting Real on Real Platforms

EXECUTION ON TESTBEDS
WSN Design and Development

Simulation
- TOSSI M [Levis2003]
- PowerTOSSI M [Shnayder2004]
- Avrora [Titzer2005]

Virtualization and Emulation
- EmStar [Ganesan2004]
- BEE [Chang2003, Kuusilinna2003]

Test Grids
- moteLab [Werner-Allen2005]
- Emstar arrays [Cerpa03/04]
- Kansei [Dutta2005]

Specialized simulation tools for WSN applications
Fast-prototyping in a controlled environment
Closing in on the “real” experience

Figure abridged from D. Estrin/J. Elson
Code Distribution Using Fixed Testbed

• Traditional test grid
  – Wired backchannel
  – Programming and logging from a remote station
  – Simple centralized control and data collection

• MoteLab @ Harvard
  – Ethernet-based
  – Multi-user arbitration
  – ~200 nodes
  – Power profiling using TMO equipment

[Werner-Allen2005]
Most Current Tools are Not WSN Specific

**Simulation**
- TOSSIM [Levis2003]
- PowerTOSSIM [Shnayder2004]
- Avrora [Titzer2005]

**Virtualization and Emulation**
- EmStar [Ganesan2004]
- BEE [Chang2003,Kuusilinna2003]

**Test Grids**
- moteLab [Werner-Allen2005]
- Emstar arrays [Cerpa03/04]
- Kansei [Dutta2005]
Most Current Tools are Not WSN Specific

**Virtualization and Emulation**
- EmStar [Ganesan2004]
- BEE [Chang2003, Kuusilinna2003]

**Simulation**
- TOSSIM [Levis2003]
- PowerTOSSIM [Shnayder2004]
- Avrora [Titzer2005]

**Test Grids**
- moteLab [Werner-Allen2005]
- Emstar arrays [Cerpa03/04]
- Kansei [Dutta2005]

**Deployment**
- In-network reprogramming [Levis2004, Hui2004]
- Calibration and Verification [Szewczyk2004]
- Dependence on infrastructure [Szewczyk2004]
Remote Programming vs. Testbeds

• Remote Programming (in-band)
  - A versatile tool
  - Requires a functional runtime and communication infrastructure
  - Non-reversible
  - Depending on wireless performance
  - Limited feedback capability

• Wired Testbeds (out-of-band)
  - Lack of mobility, scalability
  - Communication bottleneck at the sink
  - No interaction with nodes
  - Cannot grow with the application
A Testbed for WSN - Deployment-Support Network

**DSN Testbed Key Differentiators**
- Distributed observers
- Mobility: Wireless, battery powered

**DSN Testbed Functionality**
- Remote reprogramming
- Extraction of log data
- Stimuli, e.g. fault injection
- Time synchronization

Target Sensor Network


- Centralized logging
- Detailed behavioral analysis
DSN - A Distributed Testbed with Local State

Device under Test

Testbed

Distributed, Stateful Observer
Extracting State in a Coordinated Way

Node-level software instrumentation

- `printf()` vs. timing
- Buffer space
- Data throughput
- Synchronization
- Compression
- Real-time integration
Other Ways of Catching Context

PASSIVE INSPECTION
Packet Level Overhearing – Packet Sniffing

- Many symptoms are detectable by passive
  - Dead nodes, Node reboots, Network partitions
  - Approximate neighborhood
- The causes of the problems remain unclear

[Ringwald DCOSS 2007, Roemer IPSN 2009]
Simple Evaluation Framework

- Great for an overview

- Limits
  - Locality
  - Non-interactive
  - Limited context
  - Wireless data loss

- Similar alternative
  - Field inspection tools [L. Selavo 2007/2008]

[L. Selavo 2007/2008]
[Ringwald DCOSS 2007]
Evaluation With Passive Distributed Assertions

- Assertions: express believe that a condition holds

- Assertions over distributed program variables
  - Value of variable i should equal value of variable k at node 100

- Checked by passive inspection
  - Nodes only broadcast assertions (PDA msg), changes of relevant variables (SNAP msg)
  - Overheard by sniffer network
  - Minimize interference

```c
int i;
...
assert(i > 50);
```

```c
int i;
...
PDA(i = 100:k);
```

```c
k = ...;
SNAP(k);
```
Assertion Evaluation

• Backend evaluates trace
  - Reconstruct variable values from SNAP
  - Evaluate PDA on variable values

• Key challenge: incomplete information
  - Missing messages
  - Inaccurate synchronization

• Goal: avoid wrong evaluation results
  - Indicate undecidable assertions to users

[Roemer IPSN 2009]
Assertion Evaluation Example

Node 1

Latest snapshot of k on node 2 before T=10

Latest snapshot of k on node 2 before T=10

T=10  i = 5; PDA(i = 2:k)

Earliest snapshot of k on node 2 after T=10

Message Trace

N=2  S=1  T=6  SNAP(k)  k=4

T=10-\Delta

N=2  S=2  T=9  \times SNAP(k)  k=5

T=10+\Delta

N=2  S=3  T=11  SNAP(k)  k=4

N=1  S=1  T=10  PDA(i=2:k)  i=5

Earliest snapshot of k on node 2 after T=10

Node 2

T=6  k = 4; SNAP(k)

T=9  k = 5; SNAP(k)

T=11  k = 4; SNAP(k)

Timesync accuracy ± \Delta
Taking Control of Resources and the Environment

PHYSICAL EMULATION
A Simple Power Trace

Target Current Consumption with Fixed Voltage Supply at 3.3V (profiled by CruiseControl)
Maybe Not So Simple?

Measurement Cycle Class F, LEDs enabled

mA

Crackmeter (build 292)
Earthpressure & Digital (build 290)
Sensorrod (build 291)
Testplug (build 293)
Current Is Looking Good!

Vcc_min = 2.8 V
A Single Snapshot Is Not Enough!!
Today's WSN Design and Development

Simulation
- TOSSIM [Levis2003]
- PowerTOSSIM [Shnayder2004]
- Avrora [Titzer2005]

Virtualization and Emulation
- EmStar [Ganesan2004]
- BEE [Chang2003, Kuusilinna2003]

Test Grids
- moteLab [Werner-Allen2005]
- Emstar arrays [Cerpa03/04]
- Kansei [Dutta2005]

Can we Emulate Reality in the Lab?

Figure abridged from D. Estrin/J. Elson
Physical Emulation Architecture

- Influence of power sources/quality
- Detailed physical characterization
- Interdependence of function and resource usage
- Injection of faults/stimuli
  - Reset of a node
  - Dropping packets, cross traffic
  - Memory corruption/overflow
- Measurements of reference sensors
- Emulation of the environment
  - Controlled RF attenuation
  - Controlled failure scenarios
  - Temperature cycling
- And of course combinations of all of the above
Integration and Automation with the DSN Testbed

[EmNets2007]
Temperature/ Humidity Cycle Testing of Software?
Physical Reality Impacts Sampling Performance

- Storage duration
- ADC duration
- Temperature
- Watchdog resets
Test Integration with the DSN Testbed

- Visualizing long-term evolution, trends, failures
- Pinpointing causes
Reaching Out for “Correct by Design” (the Ultimate Goal)

AUTOMATED VALIDATION
Towards Validation of Real-world WSN Applications
Multilevel Testing Framework

- Multiple levels of abstraction
  - Simulation (e.g. TOSSIM, AVRORA)
  - Emulation (e.g. EmStar)
  - Testbeds (e.g. DSN, moteLab)

- "Test Case" wraps around application code
  - Environment
  - Monitors
  - Stimuli

- Global pass/fail conditions
  - Unit testing
  - Parameter extraction, reporting
Automated Test Case Generation for DSN Testbed

- Detailed control, analysis and replay of simulation and testbed
- Developed and in-use at Siemens Building Technologies, Zug, CH
  - Protocols for high reliability wireless applications (fire alarm)

[DCOSS2007, INSS2007/2008]
Validation of Power Traces using Formal Bounds

- Assertions based on reference traces/specification
- Integrated with each build (regression testing)

\[ f_i(t) = \begin{cases} 
  a_0 + a_1 \cdot x + \ldots & \text{if } t \in [t_{i-1}, t_i) \\
  0 & \text{if } t \notin [t_{i-1}, t_i) 
\end{cases} \]

\[ f_i^-(t) = \begin{cases} 
  f_i(t) - \Delta y^+ & \text{if } t \in [t_{i-1}, t_i) \\
  0 & \text{if } t \notin [t_{i-1}, t_i) 
\end{cases} \]

\[ f^-(t) = \sum_{i=1}^{n} f_i^-(t) \]

\[ \forall \tilde{t} \in [-\Delta t, \Delta t], \forall i \in \mathbb{N} : \\
 f^-(t + \tilde{t}) = \begin{cases} 
  f^-(t^-) & \text{if } -f(t^-) + f(t+) \leq 0 \\
  f^+(t) & \text{if } -f(t^-) + f(t+) > 0 
\end{cases} \]

The upper bound \( f^+ \) follows accordingly with a bound value \( \Delta y^+ \).

[WEWSN2008, SUTC2008]
Conformance Testing using Timed Automata
Conformance Testing using Timed Automata

- Power trace
- Model of observed behavior
- UPAAL
- Model of expected behavior

- System in operation
- Expected behavior
Automatic Conversion of Power Traces to TA's
Conformance Testing

- Reachability of the Power Traces final location gives the acceptance condition

\[ \exists \pi \in \prod_{Sys\parallel PT} : s \stackrel{\pi}{\rightarrow} t \iff PT \models Sys \]

- Conversion from timed trace to state space
- Number of samples, noise on measurements
- False positives
Some Last Examples

Of Testing, Bugfixing and Learning it the Hard Way...
PermaSense - Test and Validation Facilities

- Networked testbed: Deployment-Support Network [Beutel – EWSN2007]
- Power profiling
- Power trace verification [Woehrle – FORMATS2009]
- Temperature cycling
- Sensor calibration
- Rooftop system break-in
Power Optimization - A Squeeze with Implications

- Regulator uses 17uA quiescent current
- Bypass used to shutdown regulator --> ~1uA in standby

- No Bypass increases ADC accuracy: std dev 0.8844 --> 0.0706
The Result - Power Quality Increases Data Accuracy

**Before**

- Crack extension
- Temperature

**After**

- Crack extension
- Temperature

std dev = 24.0 μm

std dev = 1.0 μm
But Nothing Can Replace Real Field Experience

- System Outages
- Lost Packets
- Battery Voltage & Temperature
Node Health Data - Indispensable for Analysis

- System Outages
- # Lost Packets
- Network Disconnect
- Nodes Reset
Understanding the Exact Causes is Hard...

- Missing EMP protector – broken radio chip?
- If so is it snow/wind or lightning induced?
- Long-term software bug – buffer overflow; timing?
- Misunderstood “feature” of the implementation?
- Hardware breakdown?
- Energy-supply dependent?
- Spring-time behavior based on the environment?
- Cosmic rays at 3500 m a.s.l. ?
- Can external effects trigger the reed contact on the reset?
- ...

**Task queue overrun due to timing misbehavior.**

Probably not feasible to test/ reproduce this in the lab!
Methodology and Development Tools

Testbeds

Physical Emulation

Compilation and Analysis

Execution on Real Platforms
- Distributed, native execution
- Influence of the environment
- Remote reprogramming
- Stimuli and log file analysis
- Validation

Extending the Logical View
- Detailed physical characterization
- Control of the environment
- Physical stimulation
- Control of resources

Advanced Software Engineering
- Best practices in enterprise-level SW development
- Regression (unit) testing
Test and Validation - Outlook

• **Fundamental differences** in networked embedded systems require novel approaches
  - Unreliable wireless medium
  - Distribution nature
  - Tight embedding in the environment

• Relation of functional and non-functional properties

• Thin line between function, conformance and performance

• Recent focus on formalization of our methods
  - E.g. by using Uppaal for trace analysis

• Large quantity of data requires automation and tools
A Vision of Future WSN Design Methodology...

- (Live) back annotation into the design space
  - „Closed loop“ system design
  - Including live data from simulation, testbeds, deployments

- Allowing to refine and check architectural decisions, models, algorithms and implementations...
Acknowledgements

- **BTnode core team**
  - Matthias Dyer, Oliver Kasten, Kay Roemer, Matthias Ringwald

- **PhD students**
  - Matthias Woehrle, Andreas Meier, Matthias Keller

- **PermaSense/Swiss-Experiment collaboration**
  - ETHZ, EPFL, Uni Basel, Uni Zurich, University Paderborn, SLF, Art of Technology

- **Funding**
  - SNSF (NCCR MICS), FOEN, CCES/Microsoft Research (Swiss-Experiment)

- **Further information and publications**
  - http://www.tik.ee.ethz.ch/~beutel
  - http://www.permasense.ch