Resource Management Aspects for Sensor Network Software

Sean Walton
Eric Eide

School of Computing
University of Utah

PLOS 2007
Aspect nesC

Augmenting nesC with AOP technology:

✔ To facilitate resource management in sensor network devices
✔ To experiment with AOP in embedded, componentized programming environments
✔ To aspectize embedded OS like TinyOS

Work in progress...
Limited Resources Constrain Development

Tight microcontroller (μC) resources

Sensor networks used for monitoring environments

Careful code development

<table>
<thead>
<tr>
<th>Model</th>
<th>Name</th>
<th>Bits</th>
<th>Flash</th>
<th>EEPROM</th>
<th>RAM</th>
<th>Regs</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSP430</td>
<td>F2001</td>
<td>16</td>
<td>1K</td>
<td>256</td>
<td>128</td>
<td>16</td>
</tr>
<tr>
<td>MSP430</td>
<td>F2011</td>
<td>16</td>
<td>2K</td>
<td>256</td>
<td>128</td>
<td>16</td>
</tr>
<tr>
<td>MSP430</td>
<td>F2121</td>
<td>16</td>
<td>4K</td>
<td>256</td>
<td>256</td>
<td>16</td>
</tr>
<tr>
<td>MSP430</td>
<td>F2131</td>
<td>16</td>
<td>8K</td>
<td>256</td>
<td>256</td>
<td>16</td>
</tr>
<tr>
<td>MSP430</td>
<td>F2252</td>
<td>16</td>
<td>16K</td>
<td>256</td>
<td>512</td>
<td>16</td>
</tr>
<tr>
<td>MSP430</td>
<td>F2272</td>
<td>16</td>
<td>32K</td>
<td>256</td>
<td>1024</td>
<td>16</td>
</tr>
<tr>
<td>AT</td>
<td>tiny13</td>
<td>8</td>
<td>1K</td>
<td>64</td>
<td>64</td>
<td>32</td>
</tr>
<tr>
<td>AT</td>
<td>tiny26</td>
<td>8</td>
<td>2K</td>
<td>128</td>
<td>128</td>
<td>32</td>
</tr>
<tr>
<td>AT</td>
<td>mega48</td>
<td>8</td>
<td>4K</td>
<td>256</td>
<td>512</td>
<td>32</td>
</tr>
<tr>
<td>AT</td>
<td>mega8</td>
<td>8</td>
<td>8K</td>
<td>512</td>
<td>1024</td>
<td>32</td>
</tr>
<tr>
<td>AT</td>
<td>mega169P</td>
<td>8</td>
<td>16K</td>
<td>512</td>
<td>1024</td>
<td>32</td>
</tr>
<tr>
<td>AT</td>
<td>mega32</td>
<td>8</td>
<td>32K</td>
<td>1024</td>
<td>2048</td>
<td>32</td>
</tr>
<tr>
<td>AT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Background: nesC and TinyOS

nesC is a componentized version of C.

TinyOS application comprises tasks and events.
Background: Aspect-Oriented Programming

Modularizes crosscutting concerns (CCCs)

```plaintext
module TornadoDetector {...} implementation
{ task void processSamples(Buffer* buffer) {...} }

module Diagnostic {...} implementation
{ command void processSamples(Buffer* buffer) {...} }

aspect module MyAspect {...} implementation {
  before(): execute(task void %.processSamples(%) )
  { analyzeStack(); }
  inline int analyzestack() {...}
  int MyClass.localMethod() {...}
}
```

Advice can augment, modify, or replace code.
Example Model: Tornado Detector

- Collect infrasound audio samples
- Process audio data and compare with tornado signature database
- Fourier Transform (FFT) spectral processing
- Corroborate and determine vector
Programming
Strains Capabilities

- FFTs → CPU- and stack-intensive
- Sample region (RAM) limit total sample size
- Signature database (Flash) limit sample types
- Data processing can overrun data collection
Target Example Resources

Real-time deadlines
- Event-/interrupt-driven processing requires on-time completion

Limited stack space
- μC have fixed-position/small execution stacks

API-managed resources
- Management crosses application/OS boundaries.
Real-Time Deadlines Advice

- TinyOS tasks are run-to-completion
- Atomic sections can present problems

```cpp
aspect module WatchRealtimeDeadline {
} implementation {
  advice after(): task_deadline_reached() {
    //--- Change FFT accuracy, reducing calculations
    //-- or-- Change data-sample duration
  }
  advice before(): task_completion() {
    //--- If ends before deadline
    //--- Increase analysis accuracy
    //--- If accuracy tuning was visited before
    //--- Revert accuracy
  }
}
```
Stack Overflow Advice

- Sample size determines stack usage ($O(\ln_2 N)$)
- A diagnostic mode switch may overflow stack
- Overflow response can use intermediate results
- Overflow *prevention* is the target

```cpp
aspect module PreventStackOverflow {
} implementation {
    advice inline after(): stack_overflow_imminent() &&
    within task(task void process_data(void)) {
        //--- Report the current results of processing
        //--- Stop task and release buffers
    }
}
```
Limited Resource Advice

- More data sampled → more accurate the results.
- Buffer hand-off (dangling pointers).
- Memory pool availability changes over time.
- Sending a message requires memory.
- Message sending happens infrequently.
Advice Options

What to do with system state when interrupting program flow?

– *Fixing up algorithmic state* (e.g., locks)
– *Increasing* limited resources
– *Unwrapping* jumps to a known state (e.g., setjmp())
– *Aborting* stops a task and resets state
Integrating with nesC

- Extended nesC syntax
- Standard AOP
- Embedded AOP
- Minimal code footprint
nesC Processing

nesC File → Compile nesC Source → Generate App.c → C File
Aspect nesC Processing

1. Build Aspect Rulebase
2. Compile nesC Source
3. Weave nesC with ANesC
4. Generate App.c

ANesC File
nesC File
C File
Summary

Extending nesC with AOP capabilities to:

- Facilitate resource management in sensor networks
- Capture nesC's tasks, commands, etc.
- Reconfigure component wiring
- Used two problems in a tornado example model
- Showed ANesC's integration with nesC