Debugging Sensor Code

Doina Bucur and Marta Kwiatkowska
Computing Laboratory,
University of Oxford
Sensor networks

- Pervasive Healthcare. Body Sensor Networks

- TinyOS:
  - Modern OS and language in an embedded system
Architectures

MCU

MSP430
(Texas Instruments)

AVR
(Atmel)

ARM

Telos
TelosB / Tmote Sky
eyesIFX
TinyNode

Mica2
MicaZ
Mica2Dot
Imote2

SunSpot

Debugging Sensor Code -- D. Bucur, University of Oxford
This talk is about trustworthy sensor software on the dark side of AmI (see keynote).

- Policies upon software
  - Regulate software deployed on body sensor networks, as laws regulate a doctor’s behaviour
Software

--- **Bugs:** The horror stories
European Space Agency
Ariane 5
first launch, 1996

- Legacy Ariane 4 code
- Failure: velocity encoded overflowed 16 bits
- Autodestruct
Atomic Energy of Canada
Therac-25
radiation therapy machine
1985-87

- Patients given deadly radiation overdoses (x100)
- Race condition
Toyota Prius 2004-2005

- Hybrid engine
- Gas engine would stall at high speeds
Bugs: The horror stories

Software bugs cost an economy about 0.6% of GDP [US, 2002]

Subtle
Crash the program
Security and human issues
TinyOS tool chain

Components
(threaded, C, nesC)

Platform-specific inlined program
(sequential, C + asm)

Machine code

Deployment on sensors

inline static void
RealMainP$Scheduler
$init(void ){
  SchedulerBasicP
$Scheduler$init();
}

# 45 "tos/chips/msp430/
pins/
HplMsp430GeneralIOP.nc"
static inline void
HplMsp430GeneralIOP
$38$I$set(void ){
  * (volatile uint8_t
  *) 49U |= 0x01 << 6;
}

000001c0 l *ABS*
00000000 DAC12_OCTL
000001c2 l *ABS*
00000000 DAC12_1CTL
000001c8 l *ABS*
00000000 DAC12_0DAT
000001ca l *ABS*
00000000 DAC12_1DAT
00000122 l *ABS*
00000000 DMACTL0
00000124 l *ABS*
00000000 DMACTL1
000001e0 l *ABS*
00000000 DMA0CTL

Debugging Sensor Code -- D. Bucur, University of Oxford
A TinyOS application

Application (C, nesC)

Kernel (nesC)

Drivers

Platform-specific drivers
TinyOS 2.x applications written as **TosThreads**.

Our tool builds on SATABS [3], a generic software verification tool for ANSI C; SATABS takes specifications written as user-specified assertions of boolean conditions inserted in the code. The verification is sound (and complete for finite-state applications): The program’s state space is exhaustively explored for violations of the specification, including e.g. behaviours triggered by unexpected, but possible, events such as scrambled incoming network packets. An execution trace is returned as a bug witness, allowing the programmer to correct the fault before deploying the application.

We (i) add native support for the C TosThreads API to SATABS, (ii) implement a SATABS-readable C model of the TinyOS system calls to stand in for the OS kernel, and finally (iii) verify application and kernel model against context-aware safety specifications written as SATABS assertions. We report benchmarks on running our tool on standard applications distributed with TinyOS’s sources, and on a more complex healthcare application; we find routine violations of safety requirements in staple TinyOS code.

### The Automatic Verification of TinyOS Applications

This section presents our verification method. We first overview TinyOS and the structure of a TinyOS application, which then allows us to underline possible sources of TinyOS software bugs. Finally, we assess performance with a set of benchmarks and point to the cause and nature of the bugs found.

#### Modelling the TinyOS Kernel

- `barrier_block()`
- `barrier_reset()`
- `amRadioReceive()`
- `amRadioSend()`
- `radioGetPayload()`
- `amRadioStart()`
- `sensirionSht11_temperature_read()`
- `sensirionSht11_humidity_read()`
- `hamamatsuS10871_tsr_read()`
- `hamamatsuS1087_par_read()`

### Interface to Radio

- `amRadioStart()`
- `radioGetPayload()`
- `amRadioSend()`
- `amRadioReceive()`

### Interface to Leds

- `ledOn() / Off() / Toggle()`
- `SetLeds() / getLeds()`

### Interface to Sensors

- `sensirionSht11_temperature_read()`
- `sensirionSht11_humidity_read()`
- `hamamatsuS10871_tsr_read()`
- `hamamatsuS1087_par_read()`
Software verification

- Verification vs. simulation.
- Counterexample-guided abstraction refinement (CEGAR)
- CProver tools [cprover.org]
Bugs (Policies)

Classical case:
context-aware code not coping with contextual exceptions

Categories of bugs in context-aware, TinyOS applications

<table>
<thead>
<tr>
<th>Sensing exceptions</th>
<th>Incomplete treatment of sensing errors.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network exceptions</td>
<td>Incomplete treatment of network errors.</td>
</tr>
<tr>
<td>Interface use</td>
<td>Incorrect use of interface to kernel services.</td>
</tr>
<tr>
<td>False reasoning</td>
<td>Incorrect decision-making given a context situation.</td>
</tr>
</tbody>
</table>
## Tool run times

<table>
<thead>
<tr>
<th>Application (Threads/LOC)</th>
<th>Claim line</th>
<th>Verified?</th>
<th>Time</th>
<th>Bug: context awareness</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blink 4/64</td>
<td>66</td>
<td>yes</td>
<td>2.9s</td>
<td>-</td>
</tr>
<tr>
<td>SenseAndSend 6/347</td>
<td>79</td>
<td>no</td>
<td>32.2s</td>
<td>interface use</td>
</tr>
<tr>
<td></td>
<td>136</td>
<td>no</td>
<td>1m08s</td>
<td>sensing exception</td>
</tr>
<tr>
<td></td>
<td>146</td>
<td>yes</td>
<td>4m25s</td>
<td></td>
</tr>
<tr>
<td>PatientNode 6/439</td>
<td>172</td>
<td>yes</td>
<td>29.9s</td>
<td>(interface use)</td>
</tr>
<tr>
<td></td>
<td>254</td>
<td>yes</td>
<td>3m55s</td>
<td>(sensing exception)</td>
</tr>
<tr>
<td></td>
<td>230</td>
<td>no</td>
<td>35m07s</td>
<td>network exception</td>
</tr>
<tr>
<td></td>
<td>268</td>
<td>yes</td>
<td>2m38s</td>
<td>(false reasoning)</td>
</tr>
<tr>
<td></td>
<td>262</td>
<td>yes</td>
<td>61m12s</td>
<td>(false reasoning)</td>
</tr>
</tbody>
</table>
Thank You!

Debugging Sensor Code

Doina Bucur and Marta Kwiatkowska
Computing Laboratory,
University of Oxford
Current work

Components (threaded, C, nesC)

Platform-specific inlined program (sequential, C + asm)

Machine code

Deployment on sensors

inline static void RealMainPScheduler$init(void){
    SchedulerBasicP$Scheduler$init();
}

# 45 "tos/chips/msp430/pins/HplMsp430GeneralIOP.nc"

static inline void HplMsp430GeneralIOP$38$I$set(void)
{
    * (volatile uint8_t *) 49U |= 0x01 << 6;
}