A Design Method for Modular Energy-Aware Software

OUrsi @ OU.NL
March 31, 2015

Christoph Bockisch
(christoph.bockisch@ou.nl)
Research Overview

Software Engineering Method for Energy-Aware Systems

Tool support

Conclusion
Career

2003 – 2008 PhD studies

Dissertation: An Efficient and Flexible Implementation of Aspect-Oriented Languages

2009 – 2014 Assistant Professor for Software Composition

- Software architectures for reliability & adaptivity
- Energy-optimization for embedded systems
- Language technology for aspect-oriented programming

since 2014 Assistant Professor in Software Engineering

- Data analytics in education
- Energy-optimization in software
- Verification in concurrent systems
Research Approach

Goals
- Reliability
- Adaptability
- Energy-efficiency
- Modularity

Technologies
- Language abstractions
- Software engineering methods
- Development tools
- Execution environments

Aspect-orientation
Complex events
Adaptive embedded systems
Energy-aware components in embedded software
Debugging for AOP, Verification and concurrency
Adaptive optimization

Overview
Conclusion
Engineering Energy-Aware Embedded Software

- Common goal in software engineering: **modularity**
- Energy issues do **not respect module boundaries**
  - They are a cross-cutting concern
  - Conventional approaches cannot separate energy-related code

**Approach**: method for **systematic design of energy-aware embedded software**

- Make **resources** explicit at **component interface** (energy is one possible resource)
- Facilitate implementing energy-optimization in separate components
- **Adapt & adopt tools** to support design process

**joint work with**

---

A design method for modular energy-aware software
Project Scope

Our focus

Software controlling energy-consuming devices/resources (Printer parts, mobile device components/activities, etc.)

Modular implementation of energy-related code

Not our focus

Reducing energy consumption of program execution itself

Inventing new optimization algorithms
Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network

Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network

Overview

Method

Tooling

Conclusion
Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network

Overview

Method

Tooling

Conclusion

A design method for modular energy-aware software
Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network.
Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network
Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network
Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network
Case Study: Smart Phone

Media player on a mobile phone, streaming music over the network.
Case Study: Professional Printers

- Industrial case: Océ
- Printer has few main states (start, idle, standby, running)
- All finishers have similar states
- All finishers must be in the same state
- Otherwise, system complexity unmanageable

Problem statement
- Gluer can have hot or cold glue
- Leads to two separate running states
- Increases number of states of all finishers
- Increases complexity
Case Study: Professional Printers

- Printer is connected to many finishers
- Finisher can be connected to various printers
Resource-Aware Component Interface

- **Dedicated component model**
  - Resource ports
  - Service ports
  - Resource Utilization Model (RUM)

- **RUM defined as state chart**
  - States model stable resource usage
  - Services or internal events trigger transitions

- **Resource port**
- **Service port**

### Network Manager of a 3G Network

**CELL DCH**
- radio power ≈ 800 mW
- bandwidth ≈ 100 kbps
- [inactivity timer = \(y\)]
- disconnect
- send \& receive

**CELL FACH**
- radio power ≈ 460 mW
- bandwidth ≈ 20 kbps
- [inactivity timer = \(y\)]
- disconnect
- send \& receive

**IDLE**
- radio power ≈ 0 mW
- bandwidth ≈ 0 kbps

<table>
<thead>
<tr>
<th>States</th>
<th>Resource ports</th>
<th>Service ports</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE</td>
<td>radio power 0</td>
<td>connect</td>
</tr>
<tr>
<td>CELL FACH</td>
<td>radio power 460</td>
<td>send &amp; receive</td>
</tr>
<tr>
<td>CELL DCH</td>
<td>radio power 800</td>
<td>send &amp; receive</td>
</tr>
</tbody>
</table>

**Overview**

**Method**

**Tooling**

**Conclusion**
Case Study: Smart Phone

Media Player Application (on a smart phone)

stopped

connection [until next m seconds are buffered]

play

playing

connection

[pause

connection [until next m seconds are buffered]

stop

play

Network Manager of a 3G Network

CELL DCH

radio power ≈ 800 mW
bandwidth ≈ 100 kbps

disconnect

[inactivity timer = y]

send \lor receive

CELL FACH

radio power ≈ 460 mW
bandwidth ≈ 20 kbps

[disconnect

IDLE

radio power ≈ 0 mW
bandwidth ≈ 0 kbps

Optimizing Controller

connection

download
disconnect
connect

bandwidth

receive
send
disconnect
connect

Power Supply

Connection

Stopped

Pause

Play

Radio Power

≈ 0 mW

≈ 800 mW

≈ 460 mW

≈ 0 mW

Bandwidth

≈ 0 kbps

≈ 100 kbps

≈ 20 kbps

≈ 0 kbps

Overview

Method

Tooling

Conclusion

A design method for modular energy-aware software
Design Method for Energy-Aware Embedded Software

- Identify key properties of RUM
  - Identify functional components (existing, new)
  - Identify optimizer components
- Select most suitable optimizer components
- Analyze system resource behavior


A design method for modular energy-aware software
Design Method for Energy-Aware Embedded Software

Overview

Method

Tooling

Conclusion

Identify key properties of RUM

Identify functional components
- existing
- new

Identify optimizer components

Select most suitable optimizer components

Analyze system resource behavior

Model resource behavior

Can be tool-supported for components with existing implementation.
Purpose of RUM at design-time

- **Guarantee** liveness and safety properties for all concretizations
  → Over-approximation

- **Human-readable**
  → Abstraction must be minimal
A Formal Method for Extracting RUMs

- Counterexample-Guided Abstraction Refinement (CEGAR) [16]
- Can be applied to create RUMs for existing components

Extract RUM using CEGAR

- Initial abstraction
  - Identify maximum power consumption
  - Specify one re-entrant state
  - With power consumption <= maximum
- Example key property: *in all execution sequences, the media player consumes less than 10 J for playing 20 s of music*
  - Counter example exists in abstract model
  - This counterexample does not exist in concrete model because of time-out and IDLE state

---

CEGAR for Extracting RUMs

Overview
Method
Tooling
Conclusion

- JBCPP
- JBCPP to Uppaal
- success → final RUM

- (initial) RUM
- model check
- simulate
- counterexample
- automatically refine RUM
- relate profile to events
- spurious counterexample
- analyze profile
- time
- execute and profile
- Trepn
- real counterexample

A design method for modular energy-aware software
Tool support

- Developed JBCPP
  - Ecore-based model of Java bytecode
  - Extensible (e.g., energy/time information)
- Adapted MAGIC
  - CEGAR-implementation
  - Extract RUM from C source code
  - Optimize resulting RUM
- Adopted Trepn
  - Energy profiling Android applications
- Adopted UPPAAL
  - Compose and analyze system resource behavior
  - Simulate using model checking
Design Method for Energy-Aware Embedded Software

- Identify key properties of RUM
- Identify functional components (existing, new)
- Identify optimizer components
- Select most suitable optimizer components
- Model resource behavior
- Analyze system resource behavior

When RUM is specified formally, it can be analyzed by model-checking tools.
Analyzing System Resource Behavior with UPPAAL

- Commercially used model checker
- Model, verify, and validate timed automata

- Models are finite-state machines with numeric and clock variables (RUM)
  - Transitions react to events (invocation of provided service)
  - Create events (invoke required service)
  - Variables (can represent resource consumption)

- Key properties
  - Subset of timed computation tree logic
Analyzing System Resource Behavior with UPPAAL

- **Consistency checks:**
  only use specified services and resources

- **Liveness checks:**

- **Simulate** model to determine resource usage

- **Cannot automatically choose the best composition**
Summary

- **Iterative method** for developing energy-aware software
  - Software controlling energy-intensive hardware
  - Modular implementation of optimizations
  - Specify energy (resource) behavior at interface

- **Tool for extracting resource utilization model**
  - Based on formal method
  - Yields timed automaton

- **Analysis** of system's resource utilization

- Not shown here: Programming language support for automatic, online tracking of resource state
Future Work

• Improve energy profiling
  • Software Energy Footprint lab:
    Dedicated hardware measuring energy consumption
  • High accuracy

• Use analysis result to improve profiling automatically

• Time Performance Improvement with Parallel Processing Systems
  • Use model checker simulate system with soft real-time constraints
  • Identify bottlenecks and propose optimizations
Next Research Idea

Optimize energy consumption of execution itself

- Create extensive profile:
  - Energy consumption
  - Non-deterministic behavior, such as: thread-switching, optimization decisions, garbage collection

- Discover dependencies with data mining

- Derive heuristics for non-deterministic decisions

- Possibly develop online optimizations