JouleUnit

Claas Wilke, Sebastian Götz, Sebastian Richly

March 26th, 2013
Problem:

Software causes too much energy consumption for today’s ICT applications!
Example 1: Smart Phones

“It started to drain the battery and having loooooooong load times. This must be the worst update off all.”

(Glenn)

“Used 5% of my battery within five minutes.”

(Shawn)

„I would like to logout during the night so it doesn't kill my battery.”

(Brittany)
Example 2: Roboter

- **Power Consumption:**
  - ~20W – 40W
  - **Uptime:** ~90 min.
- **Software controls** motors and thus, **energy consumption!**
Target:

Optimization of software's energy consumption
Energy Profiling

- **Prerequisite of Optimization**
- **Many problem- and device-specific solutions exist**
  - Software- and hardware-based approaches [HSB12]
  - Focus on:
    - Desktop [LL06] [SEMN08],
    - Laptop [Fli01],
    - Server [KZL+10],
    - Smart phone [PHZ12] [VSF+12] [HSB12],
    - Robot applications [KW08].
- **But: no generic, reusable, easy-to-use solutions**
  - **Target: a generic, reusable profiling framework**
REQUIREMENTS FOR ENERGY PROFILING
Profiling Requirements

/R1/ Workload capabilities
/R2/ Profiling capabilities
/R3/ Test run coordination
/R4/ Result evaluation and presentation
Requirements

/R1/ Workload capabilities

/R1-1/ Workload execution

/R1-2/ Setup and tear down functionality

/R1-3/ Event logging
Requirements

/R2/ Profiling capabilities

/R2-1/ Energy consumption profiling
/R2-2/ Automated profiler calibration
/R2-3/ Software- and hardware-based profiling
/R2-4/ On- and off-device profiling
/R2-5/ Eased integration of new devices
/R2-6/ Parallel profiling of multiple devices
/R2-7/ Profiling of further hardware information
Requirements

/R3/ Test run coordination

/R3-1/ Coordinated profiler and workload execution

/R3-2/ Multiple runs

/R3-3/ Synchronization of workload and profiling events

/R3-4/ Synchronization of system clocks

31.03.2013
Requirements

/R4/ Result evaluation and presentation
  /R4-1/ Result evaluation
  /R4-2/ Result preservation
  /R4-3/ Result presentation
JOULE UNIT
JouleUnit
Top-level architecture

**JouleUnit Workbench**
Concepts to trigger profiling, and to preserve, export and present profiling results.

**JouleUnit Coordinator**
Concepts to deploy and execute workloads, coordinate profiling and evaluate results.

**Workload Runner**
Concepts to define use cases and data, as well as for their execution.

**Energy Profiler**
Concepts to pick power rate probes and compute energy consumption.
The workload runner

- Reuses the JUnit framework
  - Workload execution (/R1-1/)
  - Setup and tear down (/R1-2/)

- **Event logging** must be provided device/OS-specific (/R1-3/)
  - E.g., by reusing Android Log Cat
The energy profiler

- **Profiling and automated calibration** (/R2-1/, /R2-2/)  
- **On-/Off-device profiling based on deployment** (/R2-3/, /R2-4/)  
- **Abstract reusable classes**  
  - Eased integration of new devices, platforms (/R2-5/)  
- **Composite profilers** supported (/R2-6/)  
- **Further hardware profiling possible** (/R2-7/)  

---

**JouleProfiler**
- calibrate()
- startProfiling()
- endProfiling(): EnergyProfile
- logEvent()

**EnergyProfile**
- getConsumedEnergy()
- getDuration()

**PowerRate**
- getPowerRate()
- getTimeStamp()
The coordination layer

- **Encapsulates workflow for coordinated workload and profiler execution** (/R3-1/)
  - Abstract implementation provides workflow skeleton
  - Support of multiple runs (/R3-2/)  

- **Synchronization of workload and profiling events** based on timestamps (/R3-3/)

- **Automated NTP-based system clock synchronization** (/R3-4/)
The JouleUnit workbench

- **Result evaluation** (/R4-1/)
  - Computation of average results and standard deviation

- **Result preservation** (/R4-2/)
  - CSV and SQL export

- **Result presentation** (/R4-3/)
  - Eclipse-integrated runners and views
  - Based on abstract concepts
    → Reusable for different devices and OSs
CASE STUDIES
JouleUnit for Android

- **Workload runner**
  - Reuse of the JUnit runner for Android

- **Energy profiler**
  - Software-based profiling (ACPI infos from /proc)
  - Hardware-based profiling (Yokogawa WT210 power meter)

- **Coordination Layer**
  - Android-specific implementation
  - Based on the Android Debug Bridge (ADB) and Log Cat
Comparing Android apps’ energy consumption

- Definition of **benchmarks** as a set of **test cases**
  - Web browsing
  - Emailing
- **Instantiation** of the benchmarks for **existing applications**
  - [WRP+12] [WRG+13]
    - EasyBrowser, DroidSurfing, NineSky
    - K9 Mail, MailDroid, MailDroid Pro
- Profiling of **each use case 50 times** per application

→ The first approach comparing apps from usage domains!
JouleUnit for NAOs

- **Workload runner**
  - Workload execution via web service [GLR+12]

- **Energy profiler**
  - Power rates from NAO’s battery sensor (20 Hz)
  - Profiling on device, retrieval via web service

- **Coordination Layer**
  - Simple Java implementation triggering JUnit and profiler

- **Overall implementation time:** 3-4 hours
Lessons learnt

- **Eased implementation for individual devices**
  - Sensible layering of framework
  - Reuse of abstract implementations

- **Reuseable coordinator implementation**
  - Saves a lot of time
  - No worries on time synchronization
Summary

• Energy optimization of software is important

• Many profiling approaches, but no generic solutions

• Requirements analysis

• JouleUnit
  • Implementations for Android and NAOs
  • http://www.jouleunit.org/

• Further questions?
  • claas.wilke@tu-dresden.de
References


References


References


JouleUnit


Claas Wilke, Sebastian Götz, Sebastian Richly

March 26th, 2013