TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik

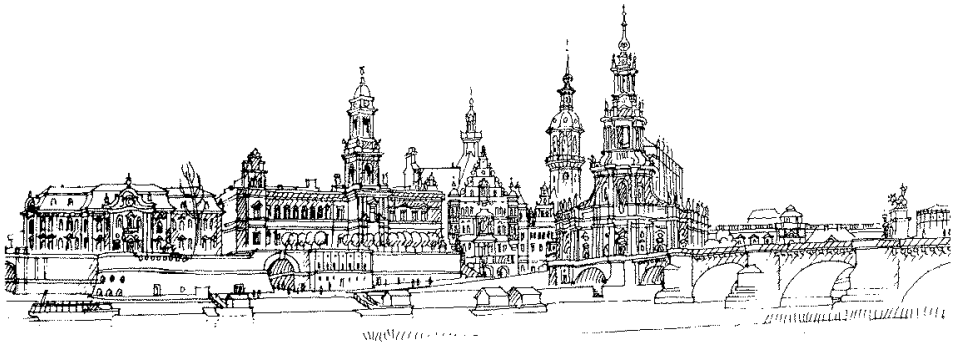
Technische Berichte
Technical Reports
ISSN 1430-211X

TUD-FI11-04-Sept. 2011

**C. Wilke,S. Götz, S. Cech,
J. Waltsgott, R. Fritzsche**

Institut für Software- und Multimediatechnik

Aspects of Software's Energy Consumption



Technische Universität Dresden
Fakultät Informatik
D–01062 Dresden
Germany
URL: http://www.inf.tu–dresden.de/

Aspects of Software's Energy Consumption

Claas Wilke, Sebastian Götz, Sebastian Cech,
Johannes Waltsgott, Ronny Fritzsche

Technische Universität Dreden, Dresden, Germany

Abstract. The CoolSoftware project focuses on optimizing software's energy consumption due to energy auto-tuning at runtime. A prerequisite for software's energy consumption optimization was to identify the different aspects that influence the energy consumption. This report summarizes the results of a literature study w.r.t. energy consumption investigations on software applications. Furthermore, it outlines the different identified aspects that have to be considered when optimizing software w.r.t. its energy consumption.

1 Introduction

The CoolSoftware[1] project focuses on building an energy auto-tuning (EAT) runtime environment for software components [14]. A central question of the project is, which aspects influence the energy consumption of software components. The major focus of this analysis comprised an intense study of existing literature, focusing on software's energy consumption and its profiling as well as its prediction. This document presents a summary of the analysis' results and gives an outlook to further research challenges and tasks. This report is related to two other reports documenting the results of similar literature analyses for hardware components [12,30].

The remainder of this report is structured as follows: Sect. 2 contains the results of our literature study focusing on current state-of-the-art solutions for energy optimization of IT infrastructures. Following, in Sect. 3, we outline the identified aspects of software application's energy consumption and how we plan to address them. Finally, Sect. 4 summarizes and concludes this report.

2 Literature Analysis

This section discusses related work w.r.t. energy consumption by hardware issued through software. The different approaches are presented in chronological order and their relation to software applications' energy consumption is shortly discussed.

In 1998 Luca Benini et al. [1] used generated workloads as input for the simulation of hardware's energy consumption. They did not focus on the application

[1] http://www.cool-software.org/

tier, but on how to compose embedded systems from hardware-components. They took software into account, but abstracted it to workloads, which they use as input for their simulation. Furthermore, they focused on single devices. They simulated state charts, whose states and transitions are weighted in terms of energy and time. Utilization-depended energy consumption was modeled using activity levels (i.e., 60% utilization of RAM). The idea has been realized and evaluated with a real world case study, showing that the simulated energy consumption is in between 5% of the actual energy consumption. Later publications often qualify this approach as inaccurate and offer other approaches for energy consumption simulation, which are more accurate. For example the cycle-accurate simulation approach of Simunic, Benini and Micheli published in 1999 [26].

Jason Flinn et al. investigated energy-efficiency of mobile devices [10]. They focused on the decision, whether a method shall be run on a mobile phone or remotely on a server. To measure resource usage, the */proc* file system and special drivers were used. The measured energy consumption was correlated with software processes and procedures. Their overall approach is termed *PowerScope* [11]. To predict resource demand Flinn et al. used mathematical models (linear and more complex), which allow to approximate a process' energy consumption for the near future.

The energy-efficient coordination of software applications on mobile devices has been investigated by Fei et al. [7,8]. Like low-power states of hardware resources, they target low-power states of software applications, based on quality of service (QoS) constraints. They specify QoS as human-perceptible characteristics of applications (e.g., for a video player these include frame rate and size). The aim of Fei's work is to energy-efficiently run multiple applications on a single mobile device, whereby the user's intension (utility) is regarded in terms of priorities set by the user.

Mahesri and Vardhan [19] performed a benchmarking case study identifying the major energy consumers of a standard laptop. They used an oscilloscope to measure energy consumed by the laptops' hardware components (hard drive, LCD/back-light, speakers, cooling fans, CPU, memory, graphics, WLAN, optical drive, modem, USB ports) either directly (i.e., cable for oscilloscope available) or subtractive (i.e., on-board components, subtracting energy consumption from total consumption) for different workloads. The workloads included standard benchmarks such as PCMark and 3DMark as well as micro-benchmarks for playing audio CDs or utilizing the network devices via FTP up- and download. Their results show that the laptop's total consumption varies a lot depending on the executed workload (i.e. software). The major consumer is the CPU—which always dominates the energy consumption if utilized. The laptop's display is another main consumer. Furthermore, the work identified that different OSs consume different amounts of energy—even for the same workloads.

Lafond et al. [18] focused on predicting the average energy consumption of Java-based applications. They profiled Java bytecode instructions by executing them on a specific JVM and a specific hardware landscape. They profiled the

energy consumption rate for a large subset of the Java bytecode instruction set and evaluated their results on several benchmarks. However, their results are only usable for a specific hardware as the bytecode's energy consumption highly depends on the hardware the Java application is executed on.

In [22] Suzanne et al. proposed an energy-aware sorting benchmark. Besides the benchmark, influences on the energy consumption of sorting algorithms are depicted. They proposed to measure the benchmarked system's energy consumption *from the wall*. Cooling systems and fans are proposed to be considered as well; AC/DC transformations were proposed to be ignored because they may consume more energy than the rest of the system.

Matthew Garrett investigated the energy consumption of hardware resources from an optimizer's point of view [13]. He points out LED displays, which can be partially dimmed to save energy, the tradeoff between frequent spin-down/-up cycles of disk drives and the lowering of their lifetime. Furthermore I/O-devices usually provide many power-modes, but to the expense of their functionality (e.g., parts of an USB-controller can be powered off, while connected devices still work, but hot-plugging of new devices is not supported). Garrett identified several causes of energy waste due to software. These comprise fixed-tick schedulers, which prohibit the CPU to scale down, because it is always in use. Drivers, waiting for hardware response by polling them are another flaw leading to unnecessary energy use. Most importantly, he presents the *race-to-idle* principle: it is best to run tasks on the system in the highest performance modes to get them done quick (race). According to Garrett, frequently switching to lower power modes rarely leads to energy savings.

In [6] Ellis provided a lecture on managing energy consumption of mobile devices. The book claims that effective power management will cut across the different levels of system design including software and hardware components. A good overview on different influences for energy consumption like hardware resources, network connections, local or remote software execution is provided. The book concludes by identifying the operating system as the appropriate layer to serve as the center of power management. Another conclusion is the statement that in general, application programs have been developed with little or no attention paid to their energy-efficiency. Thus, an application's design may have a significant impact on the hardware's energy consumption.

Chiyoung Seo et al. [24,25] compared software architecture styles for distributed software systems in regard to their energy consumption. They developed an energy estimation framework, which does not only consider energy used due to computation, but also energy used for communication. Similar to Lafond [18], Seo et al. measured how much energy was consumed by instructions on the level of virtual machines (VMs) (bytecode) and derived the total energy usage based on this data. Notably, only energy costs were taken into account. The end user's utility, and thus energy-efficiency, were not examined by their approach.

Bircher and John [2] presented an approach predicting a specific PC hardware's power consumption based on performance events of the hardware's performance counters. They profiled the system using resistors for voltage detection

and executed several workloads based on standard benchmarks for their profiling results. They built a performance model for CPU, memory, disk, chip-set and I/O. According to their own results, their model allows predicting the components' power consumption with an error of about 9%. They identified CPUs as the largest power consumers whereas the rest of the system was identified as consuming 40%–60%.

In 2007, Lachenmann et. al [17] presented their Levels approach for automated reconfiguration of nodes in a wireless sensor network w.r.t. available battery lifetime at runtime. Levels allows defining different levels of utility for application code that are used to activate or deactivate more or less functionality w.r.t. available energy at runtime. Lachemann et al. outlined that energy consumption of individual code blocks can highly differ w.r.t. the node (i.e., the hardware) on which they are executed. They further identified two types of energy consumption: hardware components' base energy consumption and further consumption depending on both the hardware's and the executed software's internal state.

In 2008, Poess and Nambiar published a benchmark for power consumption of transaction-based systems [21]. They used the TPC-C benchmark and developed a power consumption model to approximate the power consumption based on TPC-C results. Their power consumption model defines energy consumption constants for hardware components of the system under test and sums them up to the total power consumption of a benchmark run. The power consumption model includes CPUs (their energy consumption was taken from manufacturer specifications), memory (assumed consuming $9W$ per DIMM), disks (consumption taken from specification), server chassis (assumed to consume 30% of their components plus $100W$ overhead) and disk enclosures (assumed to consume 20% of their disks' consumption). The power consumption model was evaluated using similar measurement setups as described in [22]. The workload of the TPC-C benchmark was identified as a major influence on the power consumption since it utilizes the components under test. It was argued that the model is only accurate if all hardware components are utilized in a balanced way during the benchmark's run. Environmental parameters such as server room temperature were identified as further influences. The differences between estimated an measured values varied from 10% up to 25%.

Kansal et al. [15] developed the tool JouleMeter that allows predicting the energy consumption of processes running on a Windows PC. The tool can be calibrated using either the battery sensor of a laptop or an external measuring tool when running on a continuous power supply. Afterwards, JouleMeter is able to estimate the power consumption of processes by profiling their resource utilization (e.g., CPU usage and memory allocation).

A framework for the resource utilization analysis of Java applications has been built by Navas et al. [20]. It applies formal methods and control-flow analysis to predict a method's energy consumption depending on its input parameters. The framework's energy prediction is based on the results profiled by Lafond et al. [18] and is thus only applicable for a specific hardware platform.

In [23] Eric Saxe identified resource managers and resource consumers as two different points for energy-efficiency optimization. He motivated energy-efficient software by stating, that in the past only few resources existed, which where either utilized or not, but now many, heterogeneous resources comprise a system and usually only parts of them are utilized during applications' execution. This partial utilization requires an energy-aware resource management for the software being executed. He investigated energy-efficiency from three viewpoints: a *spatial* viewpoint, a *temporal* viewpoint, and a combination of both of them. The spatial viewpoint considers locality (i.e., on which core to deploy a thread or where to store data). It also contains the power states of the different resources (which resource to put into which power state). Saxe considered power states as sets of tradeoffs (e.g., performance vs. energy-efficiency). The temporal viewpoint is about scheduling, for example the ability to batch requests. Saxe proclaimed energy proportionality and showed that base energy consumption hinders this proportionality. Additionally he showed that concurrent programs do not scale in terms of efficiency (*work done / energy consumed*). Saxe identified the *evil breed* of energy wasting software: periodic time-based pollers, like Matthew Garrett [13]. *PowerTop*[2] from Intel is a tool which excels these energy wasting programs.

In [29], Tsirogiannis et al. presented results of improving energy-efficiency of database servers focusing on queries and operations commonly used in analysis and data warehousing tasks. For database energy optimization they demonstrated that the use of different CPU operators can vary the power consumption by more than 60%. Their test infrastructure's power consumption was measured for the total system at the power supply. Additionally, the voltage of the SSD's, HDD's and CPU's power supply were measured to calculate their energy consumption. A first result was that memory energy consumption seems not to vary depending on its workload. Although disks provide different operation states like idle and active, energy consumption was measured by using workload profiles, resulting in energy/utility functions. Energy consumption of SSDs was measured as proportional, while energy consumption of HDDs was measured as non-proportional. CPUs were claimed to consume 85% of the full system's power consumption when running under full workload.

Caroll et al. [4] profiled a smart phone using several micro- and macrobenchmarks utilizing the different electronic devices of the phone to derive a mathematical model of its power consumption. Their profiled electronic devices included CPU, memory, touchscreen, graphics hardware, audio, storage, and network devices. To measure the device's consumption, sense resistors where added to the phones hardware board. The results identified the following main energy consumers: graphical devices, CPU and the phone's GSM module. The results were validated by themselves using two other smart phones showing that the general assumptions for the major energy consumers seem to be accurate [4].

[2] http://www.linuxpowertop.org/

3 Aspects of Software's Energy Consumption

This section presents the different aspects that influence software applications' energy consumption identified during the literature study outlined above. The measurement and/or prediction of software components' energy consumption is a non-trivial task since software components do not consume energy themselves, but are executed on hardware that consumes energy. Nevertheless, software components and their workload influence the hardware's energy consumption as they can trigger hardware resources to switch their inner state (e.g., from *sleep* to *active*) during their execution. We identified the following aspects and influences:

1. HW resources as CPU and hard drives,
2. HW infrastructure as power supply and fans,
3. Middleware as the operating system (OS),
4. Communication as network devices,
5. Users' workloads and expected utility,
6. Software itself (i.e., code).

A detailed discussion of all the identified aspects is given below.

3.1 Hardware Resources

Hardware resources are one of the major impacts on the energy consumption of software component execution, since the software is executed on and thus, controls the hardware. The energy consumption of software can differ significantly when redeploying the software to different hardware [3,6,9,16,17,18,23,25,29]. Hence, to predict software's energy consumption, the hardware resources used by the software must be known explicitly. Hardware resources to be considered include: CPUs, RAM, hard drives, CD and DVD drives, Buses, I/O devices, graphic and sound cards, monitors (especially for laptop PCs), as well as network devices. For laptops, tablet PCs, and smart phones, graphical devices as LCD displays and touch screens are major energy consumers to be considered [4,19].

We propose to use cool component model (CCM) models and energy contract language (ECL) contracts for the modeling of explicit dependencies between software and hardware components [14].

3.2 Further Hardware Infrastructure

Besides classical hardware resources such as CPU and RAM, further hardware can be identified that consumes energy as well. These hardware devices are not required for the software's execution in a direct manner; although they fundamentally enable the complete IT infrastructure's availability. Such devices include power supply devices, AC/DC transformers, battery chargers and fans. Related work demonstrated that such devices can significantly alter the software's energy consumption [6,13,21,22].

We propose *not* to measure such energy consumers, since they can manipulate the results significantly [22]. Thus, measurements between the system under test and its power supply (AC/DC transformer) are more promising than in front of the transformer. However, probably *subtractive* measurement approaches [22] can handle such influences when no other measurement and profiling tools are available.

3.3 IT Middleware

Besides hardware resources, classical IT infrastructures have middleware software providing the overall operability of the IT infrastructure. Such middleware includes OSs, VMs, and software component containers providing services available for software components.

The services of typical OSs can cause significant energy consumption, even if no end user software services are executed at all [13,16,23] (e.g., a pull service checking every thirty seconds if a network connection is still available can cause unnecessary energy consumption by prohibiting a state transition of hardware resources from *idle* to *sleep* modes). Furthermore, different OSs can consume different amounts of energy, even if executing the same software workloads [19].

We propose to consider middleware software as *container* resources, that are required for software component's execution and to use CCM and ECL to specify these dependencies as well.

3.4 Communication

Communication between software components may cause energy consumption as well [4,24] (e.g., two components communicating in a distributed software system may communicate via network connections). Since network connections often use failure-tolerant approaches that use multiple communication paths and possible different paths for every message sent, the communication cost of software components is hard to predict and even hard to measure [27].

Indeed communication costs can be seen as costs by using hardware resources realizing the communication. We postpone the analysis of energy consumption due to different variants of communication to the end of the project or even for future work.

3.5 Load and User Settings

The users' settings and configuration of a software system influence the its energy consumption, too. Do the users want a high definition video stream or is a smaller resolution sufficient; which frame rate is required for video presentation, which bandwidth is required?

Besides user-specific settings, the load given by parameters to a software service can influence the execution time and energy consumption of a software component as well [21] (e.g., the energy consumption of a sort algorithm depends

on the given amount and their preorder in the given collection of data to be sorted). Seo et al. identified three different types of service interfaces w.r.t. energy consumption [25]:

1. Services always behaving the same w.r.t. their energy consumption,
2. Services behaving proportionally to the given parameters,
3. Services behaving unpredictable (e.g., database queries or services using caches).

Whereas the energy consumption of the first two categories of services can be predicted or simulated, the third category's energy consumption can only be computed or predicted in a heuristic or probabilistic way. Indeed, services of the third kind relate to some external/extra source of information (e.g., database or cache). Taking these extra sources into account during modeling the system allows understanding methods of the third kind as one of the first two kinds.

We propose to use *contextors* [5] to collect the users' expectations and settings and *workload description languages* to model the load of load-dependent services. Different levels of abstraction exist. The top-most layer is a user interface. Offered services can be seen as domain concepts in a domain-specific language (DSL). Business logic defines, how service requests from the user interface are translated into internal service requests. It is not uncommon to have multiple internal software layers. The lowest software layer is translating requests to the software component container. Therefore, the translation from user interface service requests down to the hardware requires DSL-transformations.

3.6 Software

Finally, the implementation of software components itself can influence the software's energy consumption. Control flow based on parameters like `if/else`-expressions or `while`-loops can significantly influence the energy consumption. Furthermore, unnecessary or unoptimized code can influence the required resources and thus, the energy consumption of software components as well. Furthermore, building software applications consuming hardware in an suboptimal way (e.g., polling a network device to often, avoiding it switching into a sleep mode) can increase the software's energy consumption as well [6,13].

We propose to describe the internal behavior of software components in an abstract way like automata or simplified control flow graphs. Probably formal methods as abstract interpretation may help to predict the software's energy behavior [20]. Further research has to investigate the best solution for software behavior descriptions w.r.t. their energy consumption. A foundation for these challenges has been done by Peter Süttner during his diploma thesis [28].

4 Summary

In this report we outlined the results on our literature study and identification of different aspects that can influence a software's energy consumption during its

execution. These aspects are: (1) hardware resources, (2) hardware infrastructure, (3) middleware (e.g., OSs), (4) communication, (5) load and user settings, and (6) the software's implementation itself. We further outlined how the plan to respect these aspects during our EAT approach using the CCM and ECL for both software and hardware modeling and the description of their dependencies.

Acknowledgments

This work emerged from research project CoolSoftware being part of the Leading-Edge Cluster CoolSilicon, which is sponsored by the Federal Ministry of Education and Research (BMBF) within the scope of its Leading-Edge Cluster Competition. It has been co-funded by the European Social Fund and Federal State of Saxony within the research project ZESSY #080951806 and the DFG within CRC 912.

References

1. Luca Benini, Robin Hodgson, and Polly Siegel. System-level power estimation and optimization. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED-98)*, pages 173–178, New York, August 10–12 1998. ACM Press.
2. W.L. Bircher and L.K. John. Complete system power estimation: A trickle-down approach based on performance events. In *IEEE International Symposium on Performance Analysis of Systems & Software (ISPASS2007)*, pages 158–168. IEEE, 2007.
3. David J. Brown and Charles Reams. Toward energy-efficient computing. *Commun. ACM*, 53(3):50–58, 2010.
4. A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, pages 21–21. USENIX Association, 2010.
5. Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *ACM Trans. Auton. Adapt. Syst.*, 1(2):223–259, 2006.
6. Carla Schlatter Ellis. *Controlling Energy Demand in Mobile Computing Devices*. Synthesis Lectures in Mobile and Pervasive Computing. Morgan & Claypool, San Rafael, CA, USA, 1 edition, 2007.
7. Yunsi Fei. *System-level Energy Analysis and Optimization of Embedded Systems*. PhD thesis, Department of Electrical Engineering, Princeton University, 2004.
8. Yunsi Fei, Lin Zhong, and Niraj K. Jha. An energy-aware framework for dynamic software management in mobile computing systems. *ACM Trans. Embed. Comput. Syst.*, 7(3):1–31, 2008.
9. Jason Flinn. *Extending Mobile Computer Battery Life through Energy-Aware Adaptation*. PhD thesis, School of Computer Science, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA, December 2001.
10. Jason Flinn, SoYoung Park, and M. Satyanarayanan. Balancing performance, energy, and quality in pervasive computing. In *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*, page 217, Washington, DC, USA, 2002. IEEE Computer Society.

11. Jason Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *WMCSA '99: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications*, page 2, Washington, DC, USA, 1999. IEEE Computer Society.
12. Ronny Fritzsche, Johannes Waltsgott, Sebastian Götz, Claas Wilke, and Sebastian Cech. State of the Art: Optimization of Energy Consumption in Storage Systems. Technical Report TUD-FI 11-05-Sept. 2011, Technische Universität Dresden, 2011.
13. Matthew Garrett. Powering down. *ACM Queue*, 5(7):16–21, 2007.
14. Sebastian Götz, Claas Wilke, Matthias Schmidt, Sebastian Cech, and Uwe Aßmann. Towards Energy Auto Tuning. In *Proceedings of First Annual International Conference on Green Information Technology (GREEN IT)*.
15. Aman Kansal and Feng Zhao. Fine-Grained Energy Profiling for Power-Aware Application Design. In *First Workshop on Hot Topics in Measurement and Modeling of Computer Systems (HotMetrics08) at ACM Sigmetrics*, Annapolis, MD, USA, 2008. Association for Computing Machinery, Inc.
16. Jeff Kramer and Jeff Magee. Towards robust self-managed systems. *Progress in Informatics*, 5:1–4, 2008.
17. A. Lachenmann, P.J. Marrón, D. Minder, and K. Rothermel. Meeting lifetime goals with energy levels. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, pages 131–144. ACM, 2007.
18. Sébastien Lafond and Johan Lilius. An Energy Consumption Model for an Embedded Java Virtual Machine. In Werner Grass, Bernhard Sick, and Klaus Waldschmidt, editors, *Architecture of Computing Systems - ARCS 2006*, volume 3894 of *Lecture Notes in Computer Science*, pages 311–325. Springer Berlin / Heidelberg, 2006.
19. Aqeel Mahesri and Vibhore Vardhan. Power Consumption Breakdown on a Modern Laptop. In Babak Falsafi and T. VijayKumar, editors, *Power-Aware Computer Systems*, volume 3471 of *LNCS*, pages 165–180. Springer Berlin / Heidelberg, 2005.
20. J. Navas, M. Méndez-Lojo, and M.V. Hermenegildo. Safe Upper-bounds Inference of Energy Consumption for Java Bytecode Applications. In *Proceedings of The Sixth NASA Langley Formal Methods Workshop*, pages 29–32, 2008.
21. Meikel Poess and Raghuanath Othayoh Nambiar. Energy Cost, The Key Challenge of Today's Data Centers: A Power Consumption Analysis of TPC-C Results. In *Proceedings of the PVLD08*, pages 1229–1240, New York, NY, USA, 2008. ACM Press.
22. Suzanne Rivoire, Mehul A. Shah, Parthasarathy Ranganathan, and Christos Kozyrakis. Joulesort: a balanced energy-efficiency benchmark. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 365–376, New York, NY, USA, 2007. ACM.
23. Eric Saxe. Power-efficient software. *ACM Queue*, 8(1):10–17, 2010.
24. Chiyoung Seo, George Edwards, Sam Malek, and Nenad Medvidovic. A Framework for Estimating the Impact of a Distributed Software System's Architectural Style on its Energy Consumption. In *WICSA '08: Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, pages 277–280, Washington, DC, USA, 2008. IEEE Computer Society.
25. Chiyoung Seo, Sam Malek, and Nenad Medvidovic. An energy consumption framework for distributed java-based systems. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, Atlanta, Georgia, USA*, New York, NY, USA, 2007. ACM Press.

26. Tajana Simunic, Luca Benini, and Giovanni De Micheli. Cycle-accurate simulation of energy consumption in embedded systems. In *in Proc. Design Automation Conf*, pages 867–872, 1999.
27. M. Stemm and R. H. Katz. Measuring and reducing energy consumption of network interfaces in hand-held devices. *IEICE Transactions on Communications, Special Issue on Mobile Computing*, 80(8), 1997.
28. Peter Süttner. Abstrakte Verhaltensbeschreibung von CCM Softwarekomponenten. Diploma Thesis, Technische Universität Dresden, March 2011.
29. Dimitris Tsirogiannis, Stavros Harizopoulos, and Mehul A. Shah. Analyzing the energy efficiency of a database server. In *SIGMOD '10: Proceedings of the 2010 international conference on Management of data*, pages 231–242, New York, NY, USA, 2010. ACM.
30. Johannes Waltsgott, Sebastian Götz, Ronny Fritzsche, Sebastian Cech, and Claas Wilke. State of the Art: Hardware Energy Management. Technical Report TUD-FI11-06-Sept. 2011, Technische Universität Dresden, 2011.