CARL VON OSSIETZKY universität OLDENBURG

2nd Workshop EASED@BUIS 2013

— Energy Aware Software-Engineering and Development —

Proceedings

edited by

Christian Bunse
University of Applied Sciences, Stralsund

Marion Gottschalk
Carl von Ossietzky University, Oldenburg

Stefan Naumann
University of Applied Sciences, Trier

Andreas Winter
Carl von Ossietzky University, Oldenburg

OLNSE Number 4/2013
April 2013

Oldenburg Lecture Notes in Software Engineering

Oldenburg Lecture Notes
on Software Engineering (OLNSE)

Carl von Ossietzky University Oldenburg
Department for Computer Science
Software Engineering
26111 Oldenburg, Germany

Content

2nd Workshop

Energy Aware Software-Engineering and Development (EASED@BUIS)

Christian Bunse
University of the Applied Sciences Stralsund
Software-Systems
christian.bunse@fh-stralsund.de

Marion Gottschalk
Carl von Ossietzky University Oldenburg
Software-Engineering
gottschalk@se.uni-oldenburg.de

Stefan Naumann
University of the Applied Sciences Trier
Environmental Campus Birkenfeld
s.naumann@umwelt-campus.de

Andreas Winter
Carl von Ossietzky University Oldenburg
Software-Engineering
winter@se.uni-oldenburg.de

Utilization of mobile and embedded devices, and thus their induced energy consumption, is constantly increasing. Reducing the energy consumption of such devices will not only improve the carbon footprint of contemporary mobile information technology usage, but will also extend the device lifetime, improve user acceptance and reduce operational costs. Next to serious and ongoing efforts in hardware design and on operating system level, software engineering techniques will also contribute to optimize energy consumption by improving software design and software quality.

The *EASED@BUIS workshop*, which follows up the *Workshop on Developing Energy Aware Software Systems (EEbS 2012)* provides a broad forum for researchers and practitioners to discuss ongoing works, latest results, and common topics of interest regarding the improvement of software induced energy consumption. We are very delighted, that BUIS-Tage 2013 in Oldenburg offered the opportunity to have *EASED@BUIS* as part of this established conference on environmental information systems. Next to providing all needed conference facilities, BUIS also allowed for an introductional paper on the development and classification of energy aware software[1] to promote our subject within their proceedings.

All together eight papers were accepted for presentation at *EASED@BUIS*. The papers were selected according their ability to invoke encouraging and fruitful discussions during the workshop. All papers were commented by 2–3 members of the program committee. The proceedings of *EASED@BUIS* will be published at Softwaretechnik-Trends.

We like to express our deepest gratitude to all authors, who submitted their thoughts to *EASED@BUIS*. We also like to thank our reviewers and sub reviewers:

- Colin Atkinson (University Mannheim)
- Paris Avgeriou (University of Groningen)
- Holger Eichelberger (University Hildesheim)
- Miguel Alexandre Ferreira (Software Improvement Group, Amsterdam)
- Sebastian Götz (TU Dresden)
- Theo Härder (TU Kaiserslautern)
- Mirco Josefiok (OFFIS, Oldenburg)
- Àkos Kiss (University of Szeged)
- Sonja Klingert (University Mannheim)
- Patricia Lago (VU University Amsterdam)
- Thierry Leboucq (KaliTerre, Nantes)
- Birgit Penzenstadler (TU München)
- Olivier Philippot (KaliTerre, Nantes)
- Giuseppe Scanniello (University of Basilicata)
- Maximilian Schirmer (Bauhaus-University Weimar)
- Gunnar Schomaker (OFFIS, Oldenburg)
- Alexandru Telea (University of Groningen)
- Joost Visser (Software Improvement Group, Amsterdam)
- Claas Wilke (TU Dresden).

Their timely comments gave reasonable help to improve the submissions. Special thanks go to the organizers of BUIS-Tage Jorge Marx Gómez, Barbara Rapp, and Andreas Solsbach for hosting our small workshop. Without their support and flexibility we would not have been able to organize *EASED@BUIS*.

Enjoy *EASED@BUIS*

Christian Bunse
Marion Gottschalk
Stefan Naumann
Andreas Winter

The third issue of *Energy Aware Software-Engineering and Development (EASED)* is taking place on Sept. 16, 2013 at GI-Jahrestagung 2013 in Koblenz. For more information, please refer to http://www.se.uni-oldenburg.de/EASED3

[1] Christian Bunse, Stefan Naumann and Andreas Winter: *Entwicklung und Klassifikation energiebewusster und energieeffizienter Software* in: Jorge Marx Gómez, Corinna V. Lang and Volker Wohlgemuth: *IT-gestütztes Ressourcen- und Energiemanagement, Konferenzband zu den 5. BUIS-Tagen*, Springer:Heidelberg, 2013.

On the Energy Consumption of Design Patterns

Christian Bunse
University of Applied Sciences Stralsund
Zur Schwedenschanze 15
18435 Stralsund

Sebastian Stiemer
University of Applied Sciences Stralsund
Zur Schwedenschanze 15
18435 Stralsund

March 29, 2013

Introduction. Energy is one of the most limiting factors for information & communication technologies in general and, more specifically for mobile devices such as Smartphones. In most application scenarios, mobile devices do not have a permanent power supply but use rechargeable batteries. Due to the increasing hardware performance and other device properties energy requirements increase further. However, software utilizes hardware and therefore directly affects the energy requirements of the entire system.

Energy-aware software development, energy-aware algorithms and energy-aware sensor substitution are only three examples for recent research that try to reduce energy requirements by optimizing the software rather than the hardware. Energy consumption is an important system property, that has already to be addressed in the early stages of development. In turn, this requires knowledge on best-practices and structures for developing energy-efficient software systems.

Following [1], patterns play many roles in software development: they provide a common vocabulary, reduce system complexity, constitute a base for building reusable software, and act as building blocks. It is a common believe that software quality increases by pattern application. But, the impact of a pattern onto properties such as performance, security or *energy consumption* is widely unknown.

In this paper, we compare the impact of design patterns onto the energy consumption of mobile (i.e., smartphone based) applications. Small apps for the Android platform were developed that either use or not use a specific pattern. The energy consumption of these apps was measured by using the PowerTutor-App, developed at the University of Michigan. The results regarding the selected pattern subset (*facade, abstract factory, observer, decorator, prototype, and template method*) are interesting. Especially the decorator pattern show a significant negative impact onto energy consumption.

Background. The research presented in this paper is rooted in the research fields of energy-aware computing and energy requirement ascertainment techniques. Many energy-aware approaches either try to reduce energy needs by substituting hardware resources [2], or by balancing energy requirements and information quality [3]. In [4] it is illustrated that a simple substitution of the resources central processing unit, and memory helps to reduce the amount of energy required. The authors of [5] showed that processing less precise data requires less energy, and also present a setup for measuring the energy requirements of core and memory of a micro controller based system, running sorting algorithms.

Energy measurement for software can either be based on hardware or software-based approaches [6]. [7] provides an approach for generating energy models for mobile systems by using the smart battery interface accompanied by means to achieve accuracy. Tools such as the Nokia Energy Profiler or PowerTutor [8] enable developers to monitor power consumption. These tools are based on an underlying cost model that, itself, is derived by analyzing a specific device (i.e. Nokia S60). [9] presents a power modeling scheme and an implementation that allows fine-grained energy accounting.

In software engineering, a pattern is a general repeatable solution to a commonly occurring problem [1]. A pattern is an abstract template that needs to be refined and adapted before it can be integrated into the code. Patterns focus on descriptions that communicate the reasons for design decisions. But, little is known about their impact onto system properties [10]. [11] examines the impact of using design patterns onto performance and provide a process for pattern selection. [12] presents an approach for mapping software design to power consumption and analyze how design decisions affect an application's energy usage. [13] analyzes six design patterns and explore the effect of them on energy consumption and performance.

Experiments Existing approaches have shown that pattern usage impacts energy consumption at least when it comes to embedded systems [13] or C++ based code [12]. The goal of our research was evaluating the impact of patterns onto the energy consumption of mobile systems that use Java. The underlying hypothesis being that using or not using a specific pattern will significantly change energy needs. Results can then be used as a starting point for further exploration in order to identify why and how the design patterns impact energy consumption. We selected a subset of the Gamma patterns (*facade, abstract factory, observer, decorator, prototype, and template method*). To evaluate the im-

pact of a single pattern we developed two, comparable applications for each pattern that either use or not use the pattern. Implementation loosely followed standard implementations available in textbooks. Energy consumption was then measured by using the PowerTutor App running on various phones (Nexus One, Galaxy SII, Transformer) whereby experiments and measurements were coordinated by a framework.

				Difference (%)	
Pattern	System	Overall Time	Energy (J)	Time	Energy
Facade	"Clean"	15,40	395,60	1,80	2,50
	Pattern	15,70	405,60		
Abstract Factory	"Clean"	13,50	342,10	14,20	15,90
	Pattern	15,40	396,60		
Observer	"Clean"	15,10	373,70	0,90	0,10
	Pattern	15,20	373,90		
Decorator	"Clean"	15,20	374,00	132,40	133,60
	Pattern	35,40	873,80		
Prototype	"Clean"	11,20	271,80	33,00	33,20
	Pattern	14,90	362,00		
Template Method	"Clean"	15,00	366,40	0,30	0,10
	Pattern	15,10	366,70		

Figure 1: Experimental Results

Table 1 shows the results of the first experiment series. While measurements for patterns such as Facade, Observer or Template Method show no difference, the results for the Prototype and Decorator show a large difference in time and energy needs (15,2 vs. 35.4 and 374 vs. 873,8). The reason for the gross difference might be the large amount of objects instantiations and method calls of the pattern-based system. This supports the findings of [4] that memory consumption using the heap as well as the garbage collector are energy-intensive operations that also have a negative impact onto performance. Although interesting, our results can only be used as an indicator due to several threats to validity. Implementation and measurement might not be generalizable. This warrants further research.

Summary and Conclusions. In this paper, we presented a case study that examined the impact of design pattern application onto a systems energy consumption. Two groups of apps, either using or not using a pattern, were developed and measured. The results for a distinct subset of the Gamma patterns showed, that especially the decorator pattern has a negative impact on the energy needs of an app. Due to the low temporal resolution of the software measurement method, evaluations with a short runtime are error-prone and the used systems might not be representable. However, the interpretation of the evaluation results supports our hypothesis and justifies further research. Using patterns is not always a good idea. Their selection should not solely be based onto function and structure but also according other properties. Although the results of our study are not generalizable, the results indicate, that further research is warranted that examines the impact of patterns regarding different platforms and applications. Results can then be used for meta-analysis.

During our study, we were able to support our hypotheses but, in turn, also identified issues that warrant further research. First, the robustness of our approach regarding the hardware platform has to be evaluated. Furthermore, it is interesting to take a deeper look into the characteristics of the energy requirements of other patterns or idioms. Results might then be used to define anti-patterns regarding software energy consumption.

References

[1] E. Gamma, R. Helm, R. E. Johnson, and J. M. Vlissides, "Design patterns: Abstraction and reuse of object-oriented design," in *ECOOP*, 1993.

[2] C. Bunse and H. Höpfner, "Resource substitution with components — optimizing energy consumption," in *ICSOFT '08 Proc.*, pp. 28–35, INSTICC, 2008.

[3] J. P. Sousa, R. K. Balan, and D. G. et al, "User Guidance of Resource-Adaptive Systems," in *ICSOFT '08 Proc.*, INSTICC, 2008.

[4] C. Bunse, H. Höpfner, S. Roychoudhury, and E. Mansour, "Energy efficient data sorting using standard sorting algorithms," in *Software and Data Technologies*, pp. 247–260, Springer, 2011.

[5] H. Höpfner and C. Bunse, "Energy Aware Data Management on AVR Micro Controller Based Systems," *ACM SIGSOFT SEN*, vol. 35, May 2010.

[6] C. Bunse and H. Höpfner, "Ocemes: Measuring overall and component-based energy demands of mobile and embedded systems," in *GI-Jahrestagung*, 2012.

[7] M. Dong and L. Zhong, "Self-constructive high-rate system energy modeling for battery-powered mobile systems," in *Proc. of the 9th Intl. conference on Mobile systems, applications & services*, ACM, 2011.

[8] L. Zhang, B. Tiwana, and Z. Q. et al., "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones," in *CODES/ISSS '10 Proc.*, ACM, 2010.

[9] A. Pathak, Y. C. Hu, and M. e. a. Zhang, "Fine-grained power modeling for smartphones using system call tracing," in *Proc. of the 6th conference on Computer systems*, ACM, 2011.

[10] D. Gross and E. S. K. Yu, "From non-functional requirements to design through patterns.," *Requir. Eng.*, vol. 6, no. 1, pp. 18–36, 2001.

[11] N. Mani, D. C. Petriu, and M. Woodside, "Towards studying the performance effects of design patterns for service oriented architecture," in *Proc. of the 2nd WOSP/SIPEW intntl conference on Performance engineering*, ICPE '11, ACM, 2011.

[12] C. Sahin, F. Cayci, and I. L. M. e. a. Gutierrez, "Initial explorations on design pattern energy usage," in *1st International Workshop on Green and Sustainable Software (GREENS)*, ACM, 2012.

[13] A. Litke, K. Zotos, and E. C. et al, "Energy consumption analysis of design patterns," in *Proc. World Academy of Science, Engineering & Technology*, 2005.

Proactive Energy-Aware System Software Design with SEEP

Timo Hönig, Christopher Eibel, and Wolfgang Schröder-Preikschat
Friedrich–Alexander University Erlangen–Nuremberg
{thoenig,ceibel,wosch}@cs.fau.de

Björn Cassens and Rüdiger Kapitza
TU Braunschweig
{b.cassens,rkapitza}@tu-bs.de

1 Introduction and Motivation

Designing system software currently optimizes program code for correctness and speed. While this is essential for the reliable operation of computer systems, these two characteristics alone are often not sufficient. Moreover, it is important to ensure that a third characteristic is being considered during the process of designing system software: energy efficiency.

As optimizing program code for energy efficiency is a tedious and time-consuming task we are working on SEEP [1], a project which provides a programming framework to assist developers at the task of energy-aware programming. The framework is named after two of its key components: symbolic execution and energy profiles. In this position paper, we introduce the SEEP approach, detail our current work, and discuss future challenges. We believe that it is essential to supply software developers and software designers with the right set of tools in order to ease the process of energy-aware programming.

We have identified the current modus operandi to be hindering for energy-efficient software development. Today, developers need to analyze program code for energy hotspots manually. This task is being performed in a *reactive* manner. Program code is first being developed and afterwards being analyzed for defects with regard to unusually high energy consumption. This manual task is cumbersome for two reasons. First, the efforts required to analyze program code for energy efficiency grow exponentially with the number of program paths of the application. Second, the amount of energy consumed differs among heterogeneous hardware platforms. Developers are required to evaluate the software on various platforms which makes the task of identifying and solving energy bugs even more unappealing.

This work was partly supported by the German Research Foundation (DFG) under grants no. FOR 1508 (subproject TP2) and SFB/TR 39 (subproject C1).

With SEEP, we provide the tooling infrastructure required to overcome current limitations. We exploit symbolic execution techniques [2] for automatic analysis of program code. Combined with energy models and platform-specific energy profiles we provide energy estimates for program code to developers as early as during the time of software development.

2 Proactive Energy-Aware Programming Using SEEP

The SEEP framework (see Figure 1) is motivated by instantly providing energy estimates, which have direct influence on the development process. Hence, commonly required feedback-based code modifications after deployment can be reduced by turning the modus operandi into a *proactive* approach.

One major effort is to offer a high degree of automation, that is, requiring as little user interaction as possible. At best, no code annotations or other changes to the program under test are necessary. With regards to programming languages, developers are free in their choice and are not forced to use special energy-aware programming languages as proposed in [3].

In order to provide precise and exhaustive energy consumption estimates, program code that is potentially being executed should be incorporated into the energy estimation process. SEEP uses symbolic execution, a technique that is effective in exploring program paths automatically. This multi-path analysis ensures that energy estimates cover a program in all its facets, and as a consequence, increase the chance to unveil hidden energy hotspots. Beforehand, executables that correspond to specific code paths (so-called path entities) need to be concretely executed to extract runtime characteristics required in subsequent steps during the analysis phase.

SEEP needs to keep the complexity of the estimation process at an absolute minimum. For this purpose, the framework relies on several different energy profiles. Besides instruction profiles, which depend on

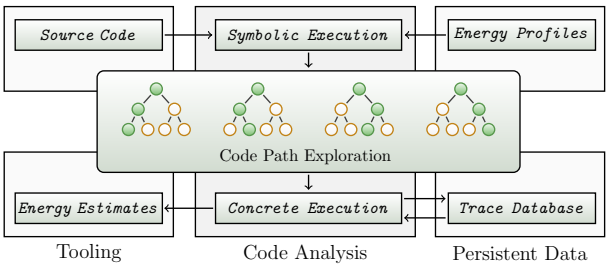

Figure 1: Overview of the SEEP architecture.

a CPU's instruction set architecture, this includes energy profiles for device-specific peripherals (e.g., network or file transfer costs). By means of virtualization environments, energy estimates can be calculated without the need to execute code on target platforms.

This profile-driven approach is extended by further persistent data, which is populated iteratively with entries for functions that have been analyzed by the framework. Such function entries consist, amongst others, of symbolic expressions, which can be exploited to interpolate a function's energy consumption. Thus, whenever the control flow of a consecutive execution run reaches functions that have been analyzed previously, symbolic execution can be omitted. This shortcut saves great amounts of analysis time.

Furthermore, concretely executing path entities to deduct a target's runtime behavior can be parallelized for heterogeneous platforms using virtualization techniques. At this, our approach does not make any restrictions as long as target platforms and their peripheral devices can be measured accurately according to a precise energy model. From a CPU's point of view, such models must contain both basic and inter-instruction energy costs which vary in dependence of a CPU's capabilities (e.g., instruction pipelining).

3 Development Process Integration

Currently, we explore different possibilities to consolidate SEEP with integrated development environments (IDEs) such as Eclipse. As basis of decision-making, energy estimates are provided at function level. These estimates are displayed in the IDE so that developers can correlate program code (i.e., source code of a function and input parameters) with energy consumption estimates.

During the development phase, code changes are being reported to the backend of the SEEP framework. Functional changes trigger a reevaluation of affected program paths. To ensure consistency the trace database is being updated incrementally. Whenever code changes cause a significant negative impact on the program code currently in development, the developer is being notified concerning this matter. If alternatives for functionally equal implementations are available (e.g., different libraries implementing the same algorithm), SEEP proposes to use the more efficient alternative. This may depend on the target platform and conditionally needs to be incorporated into the build infrastructure of the program code.

To adequately represent a multitude of energy consumption estimates (e.g., several distinct input parameters for a single function) we currently evaluate how to graphically illustrate the energy estimates within IDEs. This helps developers to easily discover energy hotspots in the program code.

4 Position Statement and Outlook

Down to the present day, program code is commonly not optimized for energy-efficiency. As developers improve their program code merely with regards to speed and correctness, it leads to the situation that system software components needlessly waste energy resources. To address this, we are convinced that new concepts for energy-aware programming need to be established. Most of all it is required to provide strong tooling support for developers to ease the task of increasing the energy-efficiency of software. Such tooling support relieves developers from manually examining software for energy hotspots by providing a high degree of automation. This is a challenging endeavor as the diversification of hardware platforms steadily increases and analyzing program code asks for high analysis efforts. In order to propagate energy-aware programming we propose SEEP, a proactive approach to address these challenges. By applying the SEEP approach, we currently increase the energy-efficiency of the Sloth operating system [4] used in the research project BATS which is founded by the German Research Foundation (DFG-Forschergruppe 1508).

References

[1] T. Hönig, C. Eibel, R. Kapitza, and W. Schröder-Preikschat. SEEP: Exploiting symbolic execution for energy-aware programming. In *Proc. of the 4th Workshop on Power-Aware Computing and Systems*, pages 17–22, 2011.

[2] C. Cadar, D. Dunbar, and D. Engler. KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In *Proc. of the 8th Symp. on Operating Systems Design and Implementation*, pages 209–224, 2008.

[3] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. Corner, and E. Berger. Eon: A language and runtime system for perpetual systems. In *Proc. of the 5th Intl. Conf. on Embedded Networked Sensor Systems*, pages 161–174, 2007.

[4] W. Hofer, D. Lohmann, F. Scheler, and W. Schröder-Preikschat. Sloth: Threads as interrupts. In *Proc. of the 30th Real-Time Systems Symp.*, pages 204–213, 2009.

Towards Predictive Self-optimization by Situation Recognition

Sebastian Götz, René Schöne, Claas Wilke, Julian Mendez and Uwe Aßmann

Fakultät Informatik, Technische Universität Dresden
sebastian.goetz@acm.org, s3970849@mail.zih.tu-dresden.de,
{claas.wilke, julian.mendez, uwe.assmann}@tu-dresden.de

Abstract: Energy efficiency of software is an increasingly important topic. To achieve energy efficiency, a system should automatically optimize itself to provide the best possible utility to the user for the least possible cost in terms of energy consumption. To reach this goal, the system has to continuously decide whether and how to adapt itself, which takes time and consumes energy by itself. During this time, the system could be in an inefficient state and waste energy. We envision the application of predictive situation recognition to initiate decision making before it is actually needed. Thus, the time of the system being in an inefficient state is reduced, leading to a more energy-efficient reconfiguration.

1 Introduction

Various approaches to self-optimizing software systems have been proposed in literature [GWCA12, CC05]. The basic principle of a feedback loop is common to all approaches. This loop comprises four steps: (1) the system continuously monitors itself and its environment, (2) it analyzes the collected data and (3) decides for or against as well as (4) performs reconfiguration. These four steps (monitor, analyze, plan/decide, act) have been introduced in literature by Oreizy et al. [OGT+99]. In our previous work [GWS+10, GWCA12], we proposed a reification of this feedback loop for energy efficiency.

For example, an audio processing system that improves the quality of audio files (e.g., noise reduction, loudness normalization, etc.) for multiple concurrent users can make good use of self-optimization in terms of energy efficiency. One strategy to save energy is to automatically choose the best server to process a user request based on the current state of the system. Imagine two servers: one is very good in terms of energy consumption but provides poor performance, and the other one is bad in terms of energy consumption but offers very good performance. Let us assume that the efficient server is currently fully utilized, whereas the inefficient server is optimally utilized but is able to process additional requests. If a user sends a request to process an audio file and agrees with waiting for the result for some time, a self-optimizing system would schedule this user request on the efficient server at a later point in time instead of the apparent choice to process the file on the currently free, but inefficient server.

The goal of self-optimizing systems is to automatically reconfigure to the most efficient state possible [ST09a]. Thus, the time of the system being in an inefficient state should be as short as possible. Figure 1 depicts the four steps of the feedback loop and highlights the actual problem: the first two steps are continuously performed until a situation, which requires reconfiguration is detected. Whenever such a situation is detected, the last two steps are performed. For the time of these two steps the system is in an inefficient state. If the system aims at optimizing energy efficiency, the time of being in an inefficient state conforms to wasted energy. A peculiarity in this regard is the question *when* the plan/decide step is performed. In our previous work [GWCA12], this step is performed when a user invokes functionality of the system. That is whenever a user interacts with the system, the optimal configuration is computed and the system gets adjusted accordingly. The approach of Chang and Collet [CC05] initiates the decision step whenever a condition, specified at design time, is sensed to be violated. The problem of both approaches is that they initiate the time-consuming decision step at a point in time, when an immediate reconfiguration is required. That is the system will be in an inefficient configuration for the time of decision making as well as the time required to reconfigure the system.

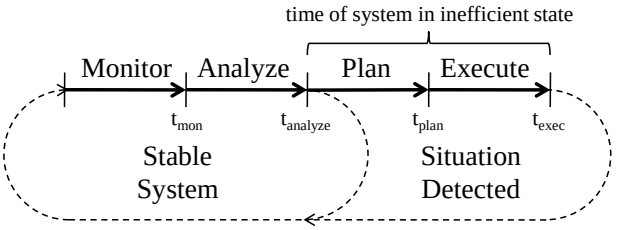

Figure 1: Time Behavior of Feedback Loop

Thus, an intelligent approach to decision making should be performed before the system gets into an inefficient state. This requires a classification of situations which demand for system reconfiguration, as it allows for the application of situation recognition to proactively detect the imminent demand for the decision step [End00]. Thereby, the time between the actual occurrence of a situation and when reconfiguration starts can, in the best case, be decreased by the time required for the decision step. In consequence, the time spans of the system being in an inefficient state are reduced.

Our envisioned solution uses predictive situation recognition [End00] based on description logic reasoning. For

this purpose we developed an automatic translation of software engineering artifacts (models, code, etc.) to the web ontology language (OWL) called OWLizer[1]. Keeping an ontology synchronous to the running software system allows the reasoner to recognize predefined situations and, by investigating the history stored in the ontology, allows for the prediction of imminent situations.

2 Envisioned Approach

As a basis for our approach to self-optimizing software system, we follow the models@run.time paradigm [MBJ+09]. The system is developed in a model-driven manner and a special model is kept synchronous to the runtime state of the system. This model is used to reason about the system. A peculiarity of our approach is the application of non-functional contracts to specify how implementations of software components behave. These contracts specify and qualify the dependencies between software components and hardware components, which are the direct consumers of energy.

To enable predictive situation recognition, we plan to synchronize the runtime model of our previous approach with an ontology using the tool OWLizer. The applicability of this synchronization has been shown in [Sch12]. The applicability of description logics for situation recognition has been shown in [ST09b]. Based on this, ontology situations can be recognized by standard reasoners. As a prerequisite, the types of situations to be recognized have to be classified. We identified two major classes of situations, which subdivide into two minor classes each:

- contract-concerned situations: (1) violation of contract clauses, which have been valid at the last decision made, (2) contract clauses, which have been invalid at the last decision made, but became valid
- user-concerned situations: (1) system overload, i.e., the system cannot serve more users, (2) increasing/decreasing number of concurrent users

Each of these situation types potentially demands for reconfiguration as they imply the likelihood of the existence of a better system configuration. For example, imagine a contract clause for an audio normalization algorithm, which specifies the requirement to have a CPU with less than 20% load to guarantee a processing time of at most 10% of the time a playback of the audio file would take. Notably, the processing time impacts energy consumption, as the longer the algorithm runs, the more energy will be consumed. The system decides to run this algorithm on a CPU with 10% load. Later, the load of this CPU increases to 50% due to some other processes, which are not under control of the self-optimizing software system. This situation is a contract clause violation. The guarantee of the contract (max. processing time) does not hold anymore, and another system configuration needs to be computed.

[1] http://st.inf.tu-dresden.de/owlizer

3 Conclusion

In this paper we first motivated the application of self-optimizing software systems to improve the energy efficiency of software at runtime. Then, we outlined the need to anticipate decision making in self-optimizing software systems to reduce their time in inefficient configurations. This is because the time of the system in an inefficient configuration implies a potential waste of energy. Finally, we proposed to utilize predictive situation recognition to detect imminent situations, which demand for reconfiguration. Thus, decision making can be anticipated and the inefficient time of the system can be reduced.

Acknowledgements

This research has been funded within the Collaborative Research Center 912 (HAEC), funded by the German Research Foundation (DFG) and within the research project ZESSY #080951806, by the European Social Fund (ESF) and Federal State of Saxony

References

[CC05] H. Chang and P. Collet. Fine-grained contract negotiation for hierarchical software components. In *31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 28–35, 2005.

[End00] M. R. Endsley. Theoretical underpinnings of situation awareness: a critical review. In M. R. Endsley and D. J. Garland, editors, *Situation Awareness Analysis and Measurement*, pages 3–32. Lawrence Erlbaum Associates, Mahwah, NJ, USA, 2000.

[GWCA12] Sebastian Götz, Claas Wilke, Sebastian Cech, and Uwe Aßmann. *Sustainable ICTs and Management Systems for Green Computing*, chapter Architecture and Mechanisms for Energy Auto Tuning, pages 45–73. IGI Global, June 2012.

[GWS+10] 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)*, pages 122–129. GSTF, 2010.

[MBJ+09] Brice Morin, Olivier Barais, Jean-Marc Jezequel, Franck Fleurey, and Arnor Solberg. Models@ Run.time to Support Dynamic Adaptation. *Computer*, 42(10):44–51, 2009.

[OGT+99] Peyman Oreizy, Michael M. Gorlick, Richard N. Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S. Rosenblum, and Alexander L. Wolf. An Architecture-Based Approach to Self-Adaptive Software. *IEEE Intelligent Systems*, 14:54–62, May 1999.

[Sch12] René Schöne. Ontology-based Contract-Checking for Self-Optimizing Systems. Minor thesis, Technische Universität Dresden, December 2012.

[ST09a] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Trans. Auton. Adapt. Syst.*, 4:14:1–14:42, May 2009.

[ST09b] Thomas Springer and Anni-Yasmin Turhan. Employing Description Logics in Ambient Intelligence for Modeling and Reasoning about Complex Situations. *Journal of Ambient Intelligence and Smart Environments*, 1(3):235–259, 2009.

Classifying Green Software Engineering - The GREENSOFT Model

Stefan Naumann, Eva Kern
Institute for Software Systems
Environmental Campus Birkenfeld
P.O. Box 1380, D-55761 Birkenfeld
(s.naumann|e.kern)@umwelt-campus.de

Markus Dick
Sustainable Software Blog
http://sustainablesoftware.blogspot.de
sustainablesoftwareblog@gmail.com

Introduction. Up to now several relationships between Information and Communication Technology (ICT) and Sustainable Development (SD) are published. However, especially in the field of energy aware or green software there is a lack of detailed descriptions. Since this field is rising, it is useful to formulate some definitions and take a look at the life cycle of software. These classifications can also help to develop a research agenda for energy aware software and its development.

Green Software and Green Software Engineering. In many cases, the reason for establishing energy efficient software and systems is to achieve a longer battery life or generally to reduce costs. On top of that, moving to the ecological part of sustainability, there is the potential to decrease energy and resource consumption by ICT to support a SD. A first impression on how software influences the life cycle of the hardware by requiring more and more resources is given by Hilty. The so called "Software Bloat" denotes the effect that the availability of more powerful hardware in the near future, relaxes software developers efforts to produce highly efficient code [Hilty 2008]. A methodology to measure and incrementally improve the sustainability of software projects is presented by [Albertao et al. 2010]. They say it is advisable to implement sustainable aspects continuously, divided into the following phases: Assessment Phase, Reflection Phase, and Goal Improvement Phase. In order to make the different sustainability issues manageable, they pointed out properties of a quality model [Albertao et al. 2010]. Based on the life cycle of software, Taina proposed metrics [Taina 2011] and a method to calculate the carbon footprint of software [Taina 2010]. To do so, he analyzed the impacts of each software development phase for a generic project. The resulting carbon footprint is mainly influenced by the development phase, but also by the way how the software is delivered and how it will be used by the customers. The main problem regarding the calculation is that detailed data is required, which is often not available. Summarizing, we have to look after the (software) product and also after the process which produces this product. In any case, the impacts on environment and sustainable development have to be considered. That leads to the following definition: "Green and Sustainable Software is software, whose direct and indirect negative impacts on economy, society, human beings, and environment that result from development, deployment, and usage of the software are minimal and/or which has a positive effect on sustainable development" [Dick et al. 2010]. Consequently, Green and Sustainable Software Engineering should produce Green and Sustainable Software in a sustainable way.

The Software Life Cycle. Fig. 1 depicts an overview for the life cycle of software and its relationship to different levels of effects. It is inspired by the effect differentiation of [Berkhout et al. 2011] who distinguish between first order effects (effects resulting directly from the product, e.g. energy consumption), second order effects (usage results, e.g. effects of dematerialization by software), and third order or rebound effects (e.g. when an energy-efficient product leads to more energy consumption in total). In Fig. 1 we distinguish between the life cycle phases development, usage, and end of life.

The GREENSOFT Model. To summarize these different aspects of Green and Sustainable software we developed the GREENSOFT Model [Naumann et al. 2011]. This is a conceptual reference model for "Green and Sustainable Software", which has the objective to support software developers, administrators, and software users in creating, maintaining, and using software in a more sustainable way. The model comprises the shown holistic life cycle model for software products, sustainability criteria and metrics for software products, procedure models for different stakeholders, and recommendations for action, as well as tools that support stakeholders in developing, purchasing, supplying, and using software in a green and sustainable manner.

Summary and Conclusions. In our paper we described a life cycle inspired view for Green Software and its engineering. At first, we have to distinguish between the process and the product itself. Regarding the product and following the model, it is necessary to specify metrics and measurements and clar-

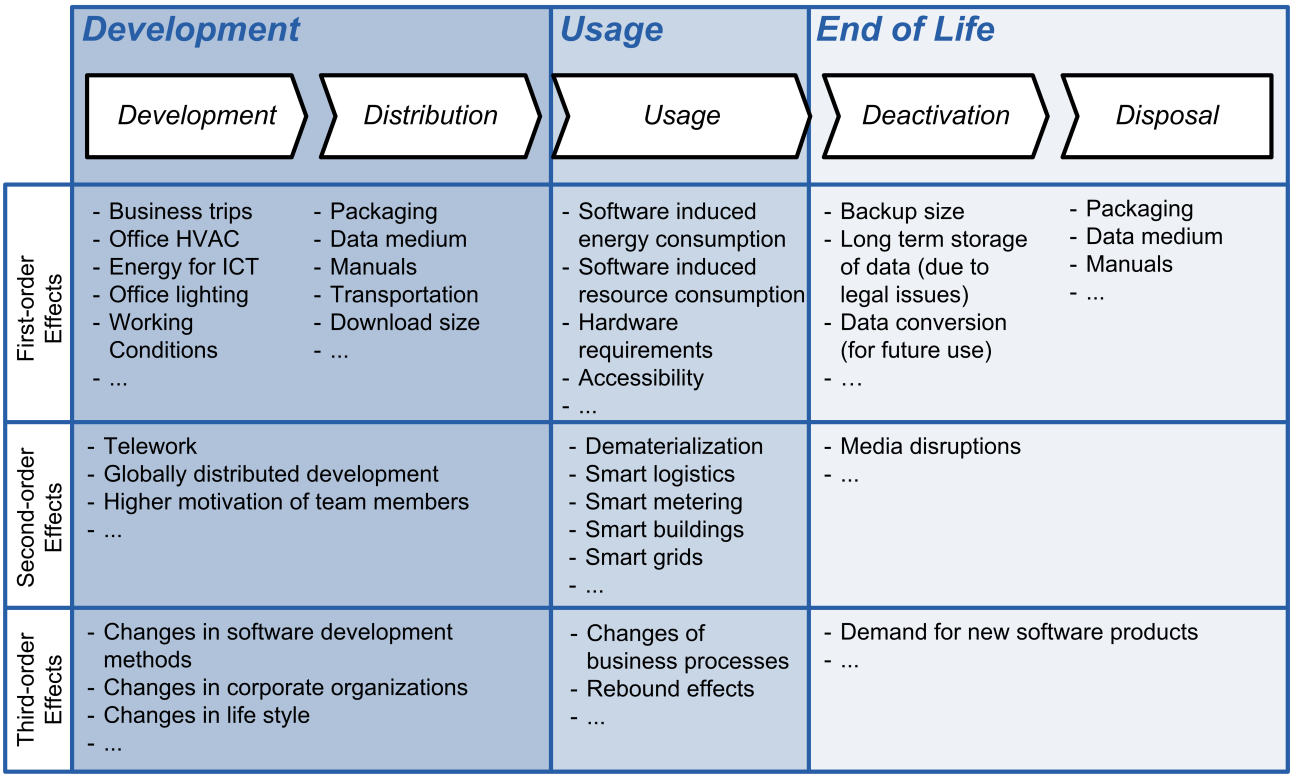

Figure 1: Software Life Cycle and Effects of different Phases (`www.green-software-engineering.de/images/downloads/green_and_sustainable_software_product_life_cycle_96dpi_web.png`), Accessed 30 March 2013

ify, how software products can be compared regarding their energy consumption. Here, it is necessary to define usage scenarios especially for standard software in order to compare different products. Another possibilty is to compare the energy consumption of different versions or releases. Here, an integration of measuring energy consumption into the continous integration process might be useful. Regarding the process, additional aspects of the sustainability of software production should be taken into account. Here, aspects like heating, greenhouse gas footprint or energy consumption have to be considered.

References

1. Albertao, F., Xiao, J., Tian, C., Lu, Y., Zhang, K. Q., and Liu, C. 2010. Measuring the Sustainability Performance of Software Projects. In 2010 IEEE 7th International Conference on e-Business Engineering (ICEBE 2010), Shanghai, China, 369-373.

2. Berkhout, F. and Hertin, J. 2001. Impacts of Information and Communication Technologies on Environmental Sustainability: speculations and evidence. Report to the OECD. `http://www.oecd.org/dataoecd/4/6/1897156.pdf`. Accessed 30 March 2013.

3. Dick, M., Naumann, S., and Kuhn, N. 2010. A Model and Selected Instances of Green and Sustainable Software. In What Kind of Information Society? Governance, Virtuality, Surveillance, Sustainability, Resilience. IFIP Advances in Information and Communication Technology 328. Springer, Berlin, Heidelberg, 248-259.

4. Hilty, L. M. 2008. Information technology and sustainability. Essays on the relationship between ICT and sustainable development. Books on Demand, Norderstedt.

5. Naumann, S., Dick, M., Kern, E., and Johann, T. 2011. The GREENSOFT Model: A Reference Model for Green and Sustainable Software and its Engineering. SUSCOM 1, 4, 294-304. doi:10.1016/j.suscom.2011.06.004

6. Taina, J. 2010. How Green Is Your Software? Software Business. First International Conference, ICSOB 2010, Jyväskylä, Finland, June 21-23, 2010. Proceedings. Lecture Notes in Business Information Processing 51, 151-162.

7. Taina, J. 2011. Good, Bad, and Beautiful Software - In Search of Green Software Quality Factors. CEPIS UPGRADE XII, 4, 22-27.

PUE for end users - Are you interested in more than bread toasting?

Kay Grosskop
Software Improvement Group, Amsterdam, The Netherlands
Email: k.grosskop@sig.eu

***Abstract*—The Power Usage Effectiveness (PUE) indicator for efficiency of data center infrastructure has been very successful. But focusing solely on PUE tends to restrict action to data center infrastructure management and in some situations even gives a perverse stimulus against optimization at the IT equipment and software levels. Despite the high relevance, no accepted metric has emerged to support optimization and allow the rating of the energy efficiency of the whole stack.**

This paper presents a metric, the *Consumption Near Sweet-spot* (CNS), that for a part can fill this gap. It captures how well the system-relative energy efficiency optimum and its utilization are aligned. A strong point is, that it allows a comparison of functionally very different services. The metric is compared to the *Fixed to Variable Energy Ratio* (FVER) metric for data centers recently proposed by the British Computer Society.

I. Introduction

The PUE is probably the single best known energy efficiency metric for computing infrastructure in use today. The PUE metric makes inefficiencies at the data center infrastructure level visible and allows decision makers to express requirement and achieved improvements with a conceptually simple, widely understood indicator.

But as the popularity of the PUE metric as a steering instrument has grown also its limitations have become more important. First, since it has not been designed to support end-to-end optimization of the whole computing stack, it cannot serve as an instrument to optimize the IT hardware and software that constitute the other main layers in a data center computing infrastructure (Figure 1). Even worse, infrastructure efficiency is typically higher if the data center runs at full capacity and hence optimizing solely for PUE values may result in a perverse stimulus to keep consumption of the IT as high as possible. Second, it does not relate the consumed energy to any useful work done in the data center. As far as concerned to the PUE, you could also operate bread toasters instead of doing any useful computation and still obtain a good efficiency rating. But the energy efficiency of an IT service should be expressed in terms of how much energy is used for a certain task that has some value for an end user like for example streaming a video or completing a monetary transaction.

Since IT equipment and Software are both interesting vectors of optimization it would be very useful to provide decision makers with an equally clear and widely applicable metric as PUE that targets the efficiency of the whole stack. However, two challenges to do so have proven to be hard to overcome:

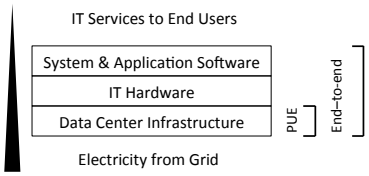

Fig. 1. The PUE indicator as part of an end-to-end service efficiency

1) Finding a unifying, yet meaningful unit of 'tasks accomplished' for functionally different applications that would allow comparison of different systems as opposed to tracking a system's energy efficiency relative to itself.
2) Defining some absolute (theoretical) optimum for the energy efficiency of a system in a similar way as it exists for the PUE, where 1 is the best possible value.

The approaches discussed in this article try to avoid these challenges. They use a system-relative definition of *unit of work* and reject the idea that a generic, normalizing unit is necessary in order to establish a useful end-to-end metric for efficiency. Moreover, instead of relating system efficiency to some absolute optimum they attempt to measure known sources of inefficiency. A well known source of inefficiency is the inability of systems to scale with load. Barroso and Hölzle have coined the term *energy proportionality* [1] to point out that efficient systems and components should be able to scale energy consumption according to the amount of work done and when idle (unutilized) they should not consume any energy at all. This is an important property to have, since many real systems are actually underutilized most of the time, are unable to scale down their power needs and hence operate generally in a very inefficient mode. (Figure 2)

Yet, many of today's most efficient systems are far from energy-proportional but instead are optimized by shaping the workload in a way that the system utilization is constantly high. Our metric is designed to acknowledge this fact and to support *two* different strategies for optimization:

1) Strive for energy proportionality.
2) Raise system utilization in order to let the system operate in an efficient load region.

The second strategy would for example use workload placement and performance tuning. The first will often boil down to reducing energy consumption for an idle or underutilized system.

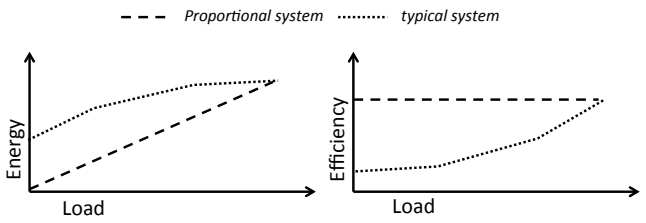

Fig. 2. Energy consumption per load level for a typical and an ideal energy-proportional system (left) and the resulting efficiency curves (right)

II. CNS

The basic idea behind our new metric can be summarized as follows. The efficiency of a system or service can be expressed by the amount of energy it takes to deliver some unit of work. This efficiency will vary over time depending on the *load level*, that is, the amount of work the system has to do at a given moment. But there will be a maximum efficiency for some system among all different load levels. This is the system-relative optimum: its *sweet spot*. It may be possible that this is still far from what is theoretically possible, but we know at least that the system can reach this efficiency in practice. A system is overall efficient if it operates most of the time close to its sweet spot.

The *consumption near sweet spot* (CNS) is computed as the ratio between the system's average consumption and its optimal consumption per unit of work. This is expressed in the following equation:

$$CNS = \frac{EU_S}{EU_{avg}} \tag{1}$$

Where EU is the energy consumed to deliver a single unit of work, i.e. the system's efficiency during a certain period. EU_{avg} is the average efficiency over an extended, representative period (e.g. a week, a month, or a year) and EU_S is the efficiency at the sweet spot, measured in a small time window when the system performs at its highest efficiency.

Since optimal consumption is always smaller than or equal to average consumption, the CNS can theoretically range between 100% (average consumption coincides with the optimum) and 0% (optimal consumption is negligible with respect to average consumption).

The metric has several strengths: It covers both of the aforementioned optimization strategies. It actually stimulates moving top efficiency and typical load regions towards each other.

Moreover it is an indicator that allows direct comparison between services. Both the type and volume of the unit of work have been fully factored out.

III. Comparison with FVER

The Data Centre Specialist Group of the British Computer Society has proposed FVER the *Fixed to Variable Energy Ratio* [2], a very similar metric to CNS although there are some important differences.

The reasoning behind FVER is that an optimal energy efficient system should behave energy-proportional. Hence the higher the variation in energy consumption at different load levels (typically between idle and max load) the more efficient it is. Energy consumption that does not vary with load (the fixed part) is suspect for being wasted.

The efficiency is expressed as a ratio between the fixed and the variable part of consumption.

$$FVER = \frac{EU_{fixed}}{EU_{variable}} \tag{2}$$

Where EU_{fixed} is the consumption when idle and $EU_{variable}$ is the difference between this idle baseline and the maximum energy consumption per unit of work. (The original formula is slightly simplified here for the sake of the discussion. Note also, that in contrast to the CNS, FVER is formulated in a way that a smaller value represents a more efficient system)

Like CNS, FVER allows for comparison of different systems by abstracting over the specific workload of a given system. It captures the concept of end-to-end energy-efficiency of a system in terms of useful work delivered to the end user.

But it has a serious shortcoming because it only takes into account the scaling behavior of the system and not the usage profile. The FVER value will not change whether I operate a system most of the time at high utilization or whether it is mostly in low utilization (and probably inefficient) mode. As such it rewards only the first of the optimization strategies mentioned in the introduction and many highly efficient systems will score low.

In contrast, CNS quantifies the extent to which the energy scaling behavior of a system matches the variability in its workload. For systems with very constant, high workload, limited scalability can already result in good CNS values. But systems that have strong fluctuations in workloads a high CNS can only be obtained with flexible scaling behavior.

IV. Future Work

We already applied the CNS metric in assessments of two industry systems. By collecting a larger number of energy profiles and CNS metrics for services across functional domains and with a wide range of workload profiles and technology footprints, we want not only get more experience in application, but also build up a register of multiple systems that will list the CNS together with two other energy efficiency indicators: the average energy consumption per unit of work and the total energy footprint of a service or system.

The CNS metric was developed together with Dirk Harryvan (Mansystems) and Jeroen Arnoldus and Joost Visser (SIG).

References

[1] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *IEEE Computer*, vol. 40, no. 12, pp. 33–37, 2007.

[2] L. Newcombe, Z. Limbuwala, P. Latham, and V. Smith, "Data centre fixed to variable energy ratio metric (dc-fver)," 2012. [Online]. Available: http://dcsg.bcs.org/groundbreaking-white-paper-new-dc-metric-available-review

An Energy Abstraction Layer for Mobile Computing Devices

Mirco Josefiok
OFFIS e.V., Oldenburg
mirco.josefiok@offis.de

Marcel Schröder Andreas Winter
Carl von Ossietzky Universität, Oldenburg
marcel.s@gmx.de winter@se.uni-oldenburg.de

Abstract

Since the growing popularity of smartphones and tablet devices, energy-efficiency in mobile computing is an increasingly interesting topic. But in case of software development engineering energy-efficiency is widely neglected, even clear and simply applicable means to measure and visualize energy consumption caused by software usage is still in its infancy. This work provides basic research in the field of measuring energy and power related information on mobile computing devices and proposes an abstract specification for implementing a measurement infrastructure on different mobile computing devices.

1 Motivation

In Germany every year more then 600 Mrd. kWh of electrical energy are consumed[1]. From this, more than 10% are accounted for information and communication technology [11]. Here, Energy consumption of mobile devices counts for ca. 11.6% [10], which results in half of the capacity of Germanys[2] most powerful nuclear power plant ISAR 2. Following the idea, that the mobile calculating power increases in the same speed, as in the past years, there is an optimization potential of about 20% to 40% of the ICT related power consumption if a holistic approach for optimizing every connected part can be found [6]. Even more, reducing energy consumption of mobil devices will also increase the lifetime of batteries due to fewer required charge cycles. But, in case of software development and engineering, improving energy-efficiency of mobile devices is widely neglected [3].

There exists many chances for optimizing energy consumption on mobile computing devices. This can be archived on different levels, ranging from hardware, operating system, and machine code to application level [4]. Low-level software optimization and improving machine code are adequately studied [8]. Optimizing energy consumption from a software point of view may vary from using alternative algorithms, resource substitution, applying compression techniques purposefully, using user profiles for energy efficient process scheduling, applying alternative hard/software-sensors, identifying and eliminating energy code smells etc. [1].

Validating these approaches and showing their contribution on saving energy requires sufficiently grained measurement techniques. Offline measurement, by using external measurement devices, is not always possible, since this requires breaking mobile devices. Online measurement, by using software means provided by the operation system, often does not furnish sufficient precision and relies on certain details of the operating system capabilities. This work aims at providing an *Energy Abstraction Layers* (EAL) in the field of mobile computing. For this purpose the EAL will abstract measurement capabilities and provide unified access to them independently from the device it is used on.

2 Requirements for an EAL

In this section the requirements for an abstract measurement specification will be described. Functional requirements resemble the intended functionality of the the EAL in an implemented form on a concrete platform. The following (most central) functional and non-functional requirements have been conducted [5]:

The EAL must provide power and energy measurement functionality. Per device and platform exists various ways of gathering energy and power related information (e.g. battery capacity, electricity and power consumption etc.). Access of those information is even restricted to certain access levels or only available an some devices.

The EAL must provide measurement functionality for estimating the runtime of each hardware component. Different hardware components in different states of operation require different energy consumption. Knowing their runtime, and combining these data with appropriate energy consumption models leads to meaningful information on hardware related energy consumption.

The EAL has to respect and report accuracy. Different measurement techniques result in different accuracy. It is the desired to know the accuracy of a measurement method [12].

The EAL should provide measurement capabilities per application and component. This would be especially helpful for finding energy wasting code patterns in applications [2].

The EAL should provide suitable error handling. As not every device and platform supports every measurement method, the EAL should report, if certain measurement capabilities are not present.

The EAL must be platform independent. For mobile devices, there exist at least two major platforms (Android, iOS) which have to be treated analogously.

[1] http://www.ag-energiebilanzen.de/componenten/download.php?filedata=1326461230.pdf&filename=BRD_Stromerzeugung1990-2011%2020Dez2011&mimetype=application/pdf

[2] http://www.kernenergie.de/kernenergie/themen/kernkraftwerke/kernkraftwerke-in-deutschland.php

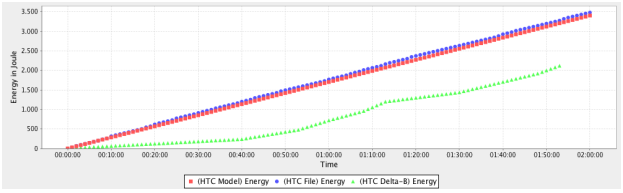

Figure 1: Energy consumption measured with three different methods on a HTC One X.

The EAL must be in device independent. Different devices offer different possibilities for measuring power consumption. Furthermore, energy efficiency of applications also depends on used hardware components.
The EAL should not rely on hardware measurement procedure. Measuring energy consumption my rely on various online techniques (cf. [7]).

3 Implementation

Most of the demanded requirements of the EAL were implemented in a prototype to measure the energy consumption on Android devices [9]. Measuring energy consumption requires to access voltage and the discharging current. While the voltage could be easily read by Android API methods, the discharging current is more complicated to access. The current EAL implementation provides three different techniques to estimate the discharging current. Figure 1 shows the energy consumption measured by these online techniques for running a simple GPS application.

The first and simplest technique is Delta-B measuring. Delta-B compares the changes of the battery charge level in relation to the battery capacity. All needed values are derived by using the `BatteryManager` class of Android API which is available since API level 5. Since this method is based on the battery charge level, Delta-B only provides 100 real measurement values, which finally only approximates for a realistic trend.

The second method reads the discharging current from the file system (File). Providing these files relies on the devices vendors, so these undocumented system files are not available on every device and updated in different time intervals . On a HTC One X the discharging current was updated every 60 seconds which is more accurate then the Delta-B method.

The third technique relies on energy profiles (Model) provided by the devices vendor. The average power consumption of different hardware components, like display in different brightness levels, enabled states for GPS, Bluetooth, CPU speed etc. are made available in an XML-file of the Android device. Android internally collects the up-time of these hardware components. A hidden part of the Android API provides methods to access these data. Comparing data measured at different times allows to calculate the enabled time and the related energy consumption in a millisecond resolution for many hardware components. This technique strongly depends on the quality of the provided energy model. Using this model based technique also allows to calculate the distribution of energy consumption related to different hardware components.

4 Summary

This paper shortly introduced the need for a standardized technique for measuring the energy consumption of apps running on mobile devices. Based on a set of requirements, an implementation of such an API was presented, which provides three different measurement techniques resulting in different accuracy. This API was already applied in [2] so show a trend of effectivity for energy smell refactorings. But further experiments on applying these techniques to measure energy consumption are required to show their reliability and to validate their benefit.

References

[1] C. Bunse, S. Naumann, A. Winter. Entwicklung und Klassifikation energiebewusster und energieeffizienter Software. In J. Marx Gómez, C. V. Lang, V. Wohlgemuth, (eds.), *IT-gestütztes Ressourcen- und Energiemanagement, Konferenz zu den 5. BUIS-Tagen*, to appear. Springer, 2013.

[2] M. Gottschalk, M. Josefiok, J. Jelschen, A. Winter. Removing Energy Code Smells with Reengineering Services. In U. Goltz et al. (eds.) *42. Jahrestagung der Gesellschaft für Informatik e.V.* LNI 208, pp. 441-455, 2012.

[3] H. Höpfner and C. Bunse. Energy Awareness Needs a Rethinking in Software Development. In *ICSOFT 2011 - Proceedings of the 6th International Conference on Software and Data Technologies*, Seville, Spain, 2011.

[4] J. Jelschen, M. Gottschalk, M. Josefiok, C. Pitu, and A. Winter. Towards Applying Reengineering Services to Energy-Efficient Applications. In R. Ferenc, T. Mens, and A. Cleve (eds.), *Proceedings of the 16th Conference on Software Maintenance and Reengineering*, 2012.

[5] M. Josefiok. *An Energy Abstraction Layer for mobile computing Devices.* Masterthesis, University Oldenburg, 2012.

[6] W. Nebel, M. Hoyer, K. Schröder, D. Schlitt. Untersuchung des Potentials von rechenzentrenübergreifendem Lastmanagement zur Reduzierung des Energieverbrauchs in der IKT, 2009.

[7] A. Pathak, Y. C. Hu, and M. Zhang. Where is the Energy spent inside my App?: Fine grained Energy accounting on Smartphones with Eprof. In *EuroSys '12: Proceedings of the 7th ACM european conference on Computer Systems*, pp 1–14. ACM, April 2012.

[8] K. Roy, M. C. Johnson. Software Design for low Power. In W. Nebel, J. P. Mermet (eds.), *Low power design in deep submicron electronics*, pp 433–460. Springer, Berlin, 1997.

[9] M. Schröder. Erfassung des Energieverbrauchs von Android Apps. Diplomathesis, University Oldenburg, 2013.

[10] L. Stobbe. Stromverbrauch von Informations- und Kommunikationstechnik in Deutschland. Technical Report BMWi, November 2008.

[11] L. Stobbe, N. F. Nissen, M. Proske, A. Middendorf, B. Schlomann, M. Friedewald, P. Georgieff, T. Leimbach. Abschätzung des Energiebedarfs der weiteren Entwicklung der Informationsgesellschaft Abschlussbericht an das Bundesministerium für Wirtschaft und Technologie. Technical report, Fraunhofer-Institut für System- und Innovationsforschung, 2009.

[12] L. Zhang, B. Tiwana, R. P. Dick, Z. Qian, Z. M. Mao, Z. Wang, and L. Yang. Accurate online Power estimation and automatic Battery Behavior based Power Model generation for Smartphones. In *IEEE/ACM/IFIP International Conference on Hardware/Software Codedesign and System Synthesis (CODES+ISSS)*, pp 105–114. ACM, 2010.

Towards Network-Wide Energy Estimation for Adaptive Embedded Systems

Patrick Heinrich
Fraunhofer Institute for Communication Systems ESK,
Munich, Germany
firstname.lastname@esk.fraunhofer.de

Abstract—**This paper discusses the next steps towards how system developers can easily and accurately evaluate the impact of their system design choices on energy consumption during the early stages of the design process. To do this, energy estimations in every phase of system development are necessary. Our research focuses on adaptive systems, where applications are activated according to the actual need.**
In this paper we present an approach which derives the energy consumption per application using a combination of energy-relevant software and hardware parameters. The aim is to create energy building blocks for applications to estimate the energy consumption of a system with multiple applications running on it. This approach utilizes the high environmental interaction of embedded systems where sensors and actors consume more energy than CPUs. The granularity of the energy estimation is the application level, due to focusing on adaptive systems.

Keywords - embedded systems, energy-efficiency, network-wide optimization, adaptive systems, automotive

I. INTRODUCTION

This paper focuses on estimating energy consumption in adaptive networked embedded systems during the early stages of design. As embedded systems become more prevalent and powerful, they are consuming more energy. Our research concentrates on the area of networked embedded systems commonly found in automobiles, aircraft and industrial systems. To save resources, these systems will be designed more adaptive in future. These systems activate their applications only when they are required. In today's luxury-class vehicles for instance, the electrical and electronic components draw up to 2.5 kW ([1], [2]). An increase of 100 W thus means that fuel consumption rises by 0.1 liter per 100 km, leading to an increase in CO2 emissions of 2.5 g per km [2]. This illustrates the considerable potential for energy savings, an aspect that must be factored in during the development process.

Embedded systems designers, such as those active in the automotive industry, are frequently given energy consumption requirements for the finished product. Because the automotive industry has especially long development cycles, hardware and software design choices must be made very early during the development process [3]. This makes it necessary to estimate the energy consumption early in the design process. Networked systems feature a wide range of technologies and topologies that significantly impact energy consumption down the road. For instance, the placement of functionality within an ECU impacts partial networking modes, which involves deactivating certain system components that are not being used. It was shown that partial networking can reduce energy consumption by as much as 30 percent [2].

Figure 1, which depicts the various phases of a typical system development process, reveals exemplarily that the energy savings potential gradually decreases over the course of development. Energy savings estimates are also imprecise, as shown by the dotted line.

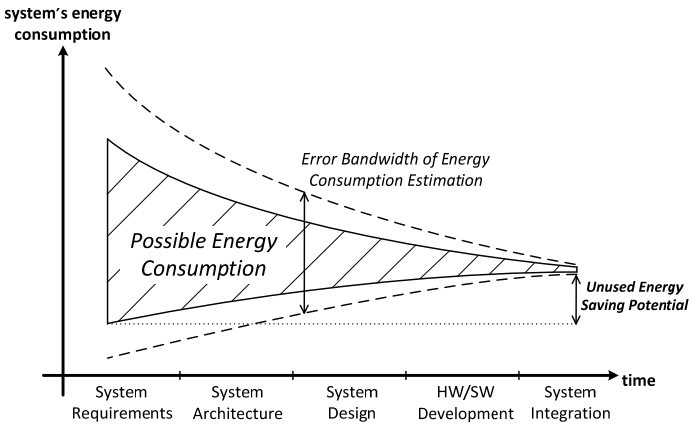

Fig. 1: Energy Design Space during System Development [4]

A structure to estimate the energy consumption of a system throughout the entire system design process is already presented in [4]. Here models for estimating the energy consumption at the different phases of development are presented which use the available system information at the particular phase.

The aim of this paper is to show the next steps towards how system developers can easily and accurately evaluate the impact of their design choices on energy consumption during the early stages of the design process. To do this, accurate energy estimations at every phase of system development are necessary. In this paper we present an approach which derives the energy consumption per application using a combination of energy-relevant software and hardware parameters. The aim is to create an energy building block per application to estimate the energy consumption of a system with multiple applications running on it.

II. ADAPTIVE NETWORKED EMBEDDED SYSTEMS

In this section, the system characteristics of adaptive networked embedded systems and the resulting challenges for estimating energy consumption are discussed.

Adaptive: Within systems such as automobiles and the context of energy saving, adaptiveness means the activation and deactivation of applications according to the current need of functionality. At the moment most automobiles support only two systems states, i.e. all on and all off, which does not utilize

the existing energy saving potentials. Future systems will have a lot more system states, because not all functionalities are necessary at every point during operation. These systems will switch their system states caused by external conditions and will be able to save energy by deactivating unused components. However, planning a lot of system states is challenging and it is unresolved how many system states optimize the energy consumption of a system [5], because also changes between system states consumes energy.

Networked: A networked embedded system consists of a number of components which communicate with each other. These components are independent processing units commonly known as Electronic Control Units (ECUs) within the automobile industry. Tasks executed on ECU depend on each other and form together applications which are visible for the user. Estimating energy consumption of networked embedded systems, for example, includes the energy demand of network communication. In addition to that, energy saving is commonly largest by deactivating whole ECUs. This is attributable to the fact that energy consumption of peripherals, memory and other parts are often not scalable – contrary to CPUs [6].

Embedded: Embedded systems are characterized by a larger amount of non-functional requirements compared to other systems such as personal computers. These limitations are for example limited resources (energy, memory, etc.) or strict time limits for the execution of tasks (deadlines). The need to design energy efficient systems makes it necessary to estimate the energy consumption as early as possible in the design process. Embedded systems are also characterized by a high degree of interaction with the environment using sensors and actors. Normally almost every application within automobiles is interacting with the environment. This results in our approach to estimate energy consumption, which is presented in the following section.

III. An Approach to Estimate Energy Consumption During System Design

In this section, an approach to derive the energy consumption per application using a combination of energy-relevant software and hardware parameters is presented. The aim is to create an energy building block per application to estimate the energy consumption of a system with multiple applications running on it. This granularity is necessary because adaptive systems activate individual applications, i.e. tasks on ECUs, according to the actual need as mentioned in section II.

To realize this, the energy consumers of embedded systems were analyzed and it was ascertained that the energy consumption of peripherals, such as sensors and actors, is much larger than the consumption of CPUs. Although CPUs are not the major consumers of energy, they are nevertheless of great importance for the energy consumption – and the software which runs on the CPU respectively. The impact of software and CPU is the resulting active time of peripherals or the whole component, because energy is defined by the used electrical power over a specific time. That means for energy estimation per application power and time estimations are necessary. We suggest deriving these information from software and hardware as outlined below.

Power Consumption: Power consumers within embedded systems are CPUs, sensors and actors and other hardware modules. Within embedded systems the major amount of power consumption is performed by sensors, actors and other application-specific hardware – not by the CPU. This results in the fact that the main part of the energy is consumed by using peripherals, i.e. when applications are active and use their specific peripherals. Applications consist of several individual tasks and the power consumption per task is assumed to be constant. This mapping of power consumption simplifies the estimation for adaptive systems, because the power consumption is assignable to a specific task, i.e. application. Through that the main part of the energy consumption of applications depends on the active time of the application, i.e. how long an application uses its peripherals.

Active Time: As discussed above the major power consumption depends on the application-specific hardware which is used by software running on the CPU. There are two causes which result the active time of an application. On the one hand there are functional requirements, for example on and off duration of indicator lights, and on the other hand the execution time of a task on a CPU. To estimate task execution times existing methods such as [7] can be used.

This approach aims energy estimation per application which can be used to estimate the energy demand of the different system states of adaptive embedded systems. Future research evaluates usability and accuracy of this approach and identifies the relevant software and hardware parameters.

IV. Conclusion and Further Steps

This paper has discussed an approach which estimates the energy consumption per application using a combination of energy-relevant software and hardware parameters. These parameters are the power consumption of application-specific hardware and the active times of this hardware resulted by software running on a CPU. This approach assigns energy demands to applications and simplifies the energy estimation for adaptive systems, because such systems activate individual applications according to the actual need. This assignment is founded on the reason that peripherals such as sensors and actors are the main consumer of energy within networked embedded systems.

Our approach allows one to evaluate the impact of the design choices on energy consumption during early stages of the design process. This enables the design of adaptive networked embedded systems which are more energy efficient. Validating our approach and evaluate the accuracy is necessary and planned for future work.

References

[1] Arthur D. Little, *Market and Technology Study Automotive Power Electronics 2015*. Available: http://www.adlittle.com/downloads/tx_adlreports/ADL_Study_Power_Electronics_2015.pdf.

[2] A. Monetti, T. Otter, and N. Ulshöfer, “Spritverbrauch senken, Reichweite erhöhen: System-Basis-Chip für den Teilnetzbetrieb am CAN-Bus,” *Elektronik Automotive*, no. 11, pp. 24–27, 2011.

[3] J. Weber, *Automotive Development Processes: Processes for Successful Customer Oriented Vehicle Development*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2009.

[4] P. Heinrich and C. Prehofer, “Early Energy Estimation in the Design Process of Networked Embedded Systems,” in *Proceedings of the 3rd International Conference on Pervasive Embedded Computing and Communication Systems*, 2013

[5] P. Heinrich and C. Prehofer, “Network-Wide Energy Optimization for Adaptive Embedded Systems,” in *Proceedings of the 4th Workshop on Adaptive and Reconfigurable Embedded Systems (APRES 2012)*, 2012

[6] R. Jejurikar and R. Gupta, “Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems,” in *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED’04)*, 2004, p. 78.

[7] P. González, P. P. Sánchez and L. Diaz, “Embedded software execution time estimation at different abstraction levels,” in *Proceedings of the XXV Conference on Design of Circuits and Integrated Systems*, 2010

Evaluation of Embedded System Energy Usage with Extended UML Models

Dmitriy Shorin and Armin Zimmermann
Ilmenau University of Technology
System & Software Engineering
P.O. Box 100 565, D-98684 Ilmenau, Germany
Dmitriy.Shorin@tu-ilmenau.de; Armin.Zimmermann@tu-ilmenau.de

***Abstract*—Energy consumption as an increasingly important decision criterion has to be included in the search for good architectural and design alternatives to make an embedded system as energy-efficient as possible. The proposed method describes a system with dedicated extended UML models for applications and hardware components and evaluates the energy use via a transformation into an analyzable stochastic Petri net.**

I. Introduction

Energy-efficiency is a so-called non-functional property which is of great importance in the current discussion about resource efficiency. While some components in the industry such as microcontrollers are already being developed with low power consumption, we should draw attention to the fact that an energy-efficient automation system as a whole includes several other components such as a controlled system and a digital control software (on which we concentrate our work for now). There is currently a lack of modeling and estimation procedures which could be applied already in the early design phases for energy consumption observation.

There are different levels of abstraction on which embedded systems can be evaluated for this task [1]. While there are some methods for the quite exact computation of energy consumption, they all require very detailed knowledge of the system under design. The description has to be on a very low level, which is available only in the later phases of the design process.

II. Method description

We propose a modeling and estimation procedure for early design phases, in which major architectural decisions are made, to consider energy consumption. The Unified Modeling Language (UML) [2] is an industry standard for the description of software systems. However, it is not intended to describe system properties equally well as there are no constructs for non-functional properties. Domain profiles of the UML have been developed for this task, namely, the MARTE Profile (Modeling and Analysis of Real-Time and Embedded Systems) [3] as a successor of the UML Profile for Schedulability, Performance and Time (SPT). With its help, non-functional properties like machine utilization, failures, temporal relations etc. can be described. The profile was developed especially for embedded systems and, hence, is suitable for our purposes better than the standard UML alone. In this work, we use state machines to describe the system behavior.

UML models adopting the MARTE profile contain the necessary information for energy consumption estimation. However, they are not usable for a numerical estimation directly, as UML models are not semantically well-defined for a specification of the resulting stochastic process. There is a lack of analysis algorithms. For this work, we propose extended UML models to be transformed into models for which analysis algorithms already exist, so that the behavior and the properties are preserved. Stochastic Petri nets (SPN) [4] are used for this purpose, as they are well adapted for concurrent and synchronized systems and powerful performance evaluation algorithms. This is an extension of an earlier work, in which extended UML statechart models were transformed into uncolored SPNs and analyzed [5]. A similar approach is taken in [6], where the work mentioned is applied to energy consumption evaluation in a different way.

In the proposed procedure, we explicitly address the hardware part of the system, which will be the same for all applications. It is described in an *operational model* and specifies all run modes of a processor (microcontroller), the possible state changes, and their associated power consumption (as well as transition times, if applicable) (Fig. 1 left top). This information can be taken from data sheets and measurements, and the model has to be constructed only once for a specific CPU.

On the other hand, the effect of the controlling software is captured in an *application model*. It describes which steps are taken and what time is spent in which mode and may include stochastic behavior (interrupts, for instance). Thus, it contains information about the

The authors would like to thank the Thuringian Ministry of Education, Science and Culture and the German Academic Exchange Service (DAAD) for the financial support of the project.

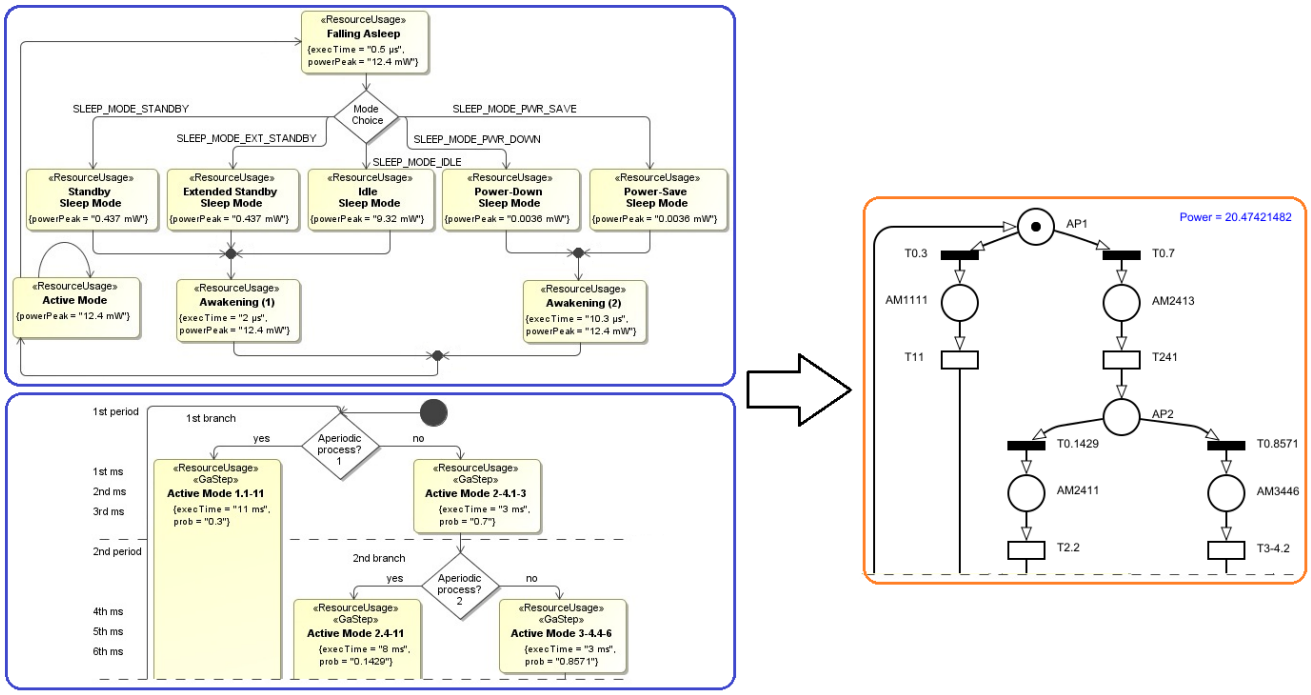

Figure 1: Operational and application UML models and their transformation into a Petri net

operational states used in the specified program and their duration (Fig. 1 left bottom).

Two created UML state machines are combined and converted into a Petri net. The application model is taken as the basic one for this operation. The missing information (missing states, power, duration) is taken from the general operational model. In [7], we presented a detailed example of the method implementation. The procedure overview is presented in Fig. 1. The resulting model can then be used to estimate the power consumption of the system with the help of Petri net tools such as TimeNET [4]. The calculation of the power occurs automatically in the course of the static analysis of the Petri net.

III. Conclusion

This paper presented a methodology for the model-based estimation of energy consumption for embedded systems. The UML language extended with the MARTE profile is used for the modeling process. The main contribution is to describe the overall processor behavior with an independent general operational model, while the software applications are specified in the application model referencing the first one. The two models are converted into a SPN, which is then used for a performance evaluation. Thus, the design process for embedded systems can be supported by predicting the energy consumption. Currently, the automatic transformation of extended UML statecharts into SPNs is being implemented as an extension of TimeNET. Besides, actual lab setups are used to apply and validate the proposed method.

References

[1] C. Talarico, J. Rozenblit, V. Malhotra, and A. Stritter, "A New Framework for Power Estimation of Embedded Systems," *Computer*, vol. 38, pp. 71–78, Feb. 2005.

[2] Object Management Group (OMG). (2011, Aug.) OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4.1.

[3] ——. (2011, Jun.) UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, Vers. 1.1.

[4] A. Zimmermann, *Stochastic Discrete Event Systems - Modeling, Evaluation, Applications.* Springer, Oct. 2007.

[5] J. Trowitzsch, "Quantitative Evaluation of UML State Machines Using Stochastic Petri Nets," Ph.D. dissertation, TU Berlin, Oct. 2007.

[6] E. Andrade, P. Maciel, T. Falcão, B. Nogueira, C. Araujo, and G. Callou, "Performance and energy consumption estimation for commercial off-the-shelf component system design," *Innovations in Systems and Software Engineering*, vol. 6, no. 1-2, pp. 107–114, 2009.

[7] D. Shorin, A. Zimmermann, and P. Maciel, "Transforming UML State Machines into Stochastic Petri Nets for Energy Consumption Estimation of Embedded Systems," in *Second IFIP Conference on Sustainable Internet and ICT for Sustainability (SustainIT 2012)*, Oct. 2012.