Towards Applying Reengineering Services to Energy-Efficient Applications

Jan Jelschen, Marion Gottschalk, Mirco Josefiok, Cosmin Pitu, Andreas Winter
Carl von Ossietzky Universität Oldenburg, Germany
{*gottschalk, josefiok, pitu, jelschen, winter*}*@se.uni-oldenburg.de*

***Abstract*—Conserving resources and saving energy has become an important issue for information and communication technology. With increasing adoption of smartphones and tablet PCs, reducing energy consumption in mobile computing is of particular significance. User expectations towards their mobile devices are rising, and functionality is increasing. Accordingly, available energy is made a scarce resource. This paper discusses how software reengineering techniques, like dynamic analysis and refactoring, can be applied to the field of energy-aware computing, to monitor, analyze, and optimize the energy profile of mobile applications and devices.**

I. Introduction

Energy-efficiency has become an important issue in information and communication technology. According to a 2007 report of German research organization Fraunhofer [1], over ten percent of Germany's overall electrical energy consumption of 2007 was generated by the information and communication technology sector. By 2020, this figure is prediced to have risen to over 20 percent, which, with unchanged energy mix, would account for CO_2 emissions exceeding that of the entire German aviation sector [2].

Energy consumption in *Mobile Computing* poses a challenge. Due to the rapidly increasing adoption of smart phones, tablet PCs, and other mobile devices, there is also an increasing demand for higher battery capacities. The most direct approach to conserve energy in IT probably is trying to build more efficient hardware. Lots of work has already been done in the area of energy efficiency in IT mostly driven by research in embedded systems and in wireless sensor networks.

Improving energy efficiency of software systems can be approached on different levels. A great amount of research focuses on low-level optimizations (e.g. machine code level) [3]. Energy efficiency for higher software layers, in particular for the software application level, has not been addressed deeply, yet. In this paper, first ideas on saving energy on the end-user application and operating system level by applying *reengineering services* are presented. Analyzing both code as well as execution behavior and patterns of applications will show opportunities for optimization. With such information, code can be restructured to utilize hardware resources more efficiently, and the operating system is enabled to schedule application execution complying with their run-time needs and a possibly system wide energy efficient behavior.

To appear in: R. Ferenc, T. Mens, and A. Cleve, Eds., *Proceedings of the 16th European Conference on Software Maintenance and Reengineering*. Los Alamitos, CA, USA: IEEE Computer Society, 2012.
©2012 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

These activities – analyzing software, then making improvements to it with the gained information and knowledge – are basically the same as in software reengineering. Reengineering aims at improving a program's quality in terms of maintainability, code quality, or similar goals [4]. This research agenda argues software reengineering tools and techniques, like *static and dynamic program analysis*, and *systematic code transformations like refactoring*, can be used to obtain more energy efficient applications.

This paper is structured as follows: Section II introduces a broad, high-level overview on the state-of-the-art of energy efficiency in IT and current and related work in that field. Section III identifies opportunities for further research. In Section IV a short overview on common and relevant reengineering services is given. Afterwards, the previously presented research opportunities are revisited in Section V, to present exemplary scenarios and ideas of applying reengineering services to energy efficiency research. Future steps towards a research agenda on energy efficient applications are sketched in Section VI. Section VII concludes this paper.

II. Energy Efficiency in IT

The rising importance of environment-friendly and energy-saving technologies also asks for the contributions software engineering and software reengineering could make to improve energy consumption. This paper originates from the results of a research seminar on graduate level to explore the field of energy-efficient applications and to identify research opportunities. The results of an initial literature review is classified along the *application domains* and the addressed *application level*, resulting in an overview on recent research into energy efficiency in information technology. For more comprehensive surveys refer to Roy and Johnson [3], who offer an overview on low-level optimization techniques, and the survey by Naik [5], covering energy efficiency in mobile computing.

A. Application Domain

Substantial ongoing research on energy efficiency in Information technology addresses four important areas:

Embedded Systems. Early research into energy efficiency has been motivated by power dissipation becoming an issue due to increasingly denser microchip designs [6], [7]. In many embedded environments this is of special importance, as sophisticated heat dissipation techniques might not be an option. Naturally, work in this area focuses mainly on hardware and low-level software optimizations.

Sensor Networks. Wireless sensor networks have been another driving force in enhancing energy efficiency (cf. [8]–[10]). These networks consist of small, autonomous devices distributed in an area, e.g. for some kind of environmental monitoring. Each single device has to last as long as possible on a single battery charge. Much work concentrates on energy-efficient, wireless data transmission, as the wireless network device is often the biggest energy consumer [11].

Data Centers. In data centers, power and cooling are becoming the dominating cost factor [12]. In this domain, optimizations therefore focus on reducing energy consumption with a fixed quality of service as goal, whereas in battery-powered devices the challenge is to deliver the best service possible with a constrained amount of energy. The recent interest in cloud computing has also spawned research into more energy-efficient data centers [13], [14]. The OFFIS Institute, associated closely with Carl von Ossietzky University, has a research group dedicated to this field (cf. e.g. Schröder et al. [15]).

Mobile Computing. With the recent advent of powerful smartphones and tablet PCs, mobile computing today addresses a large, mainstream audience. This sector has evolved very rapidly in terms of the devices' feature-richness, while development of energy-conserving techniques has not kept up. Saving energy in mobile computing will also extend the durability of battery charges, resulting in longer-dated availability of mobile devices.

When service degradation is not tolerable, energy can be saved only by making the system as a whole as *efficient* as possible in that regard. Otherwise, one can also consider trading quality of service against energy consumption. This requires the system, or parts of it, to be *energy-aware*, i.e. being able to make decisions informed by the amount of energy different strategies would require.

Notably absent from this list is *desktop computing*. While there are potentials for energy optimizations, the requirement of conserving energy is not as pressing as in mobile computing, where limited battery charge dictates energy-aware computing. Also, energy-saving measures applied on the application level will arguably be transferable from the mobile domain to desktop computers.

Whereas challenges regarding energy efficiency in embedded systems, sensor networks, and data centers need to be addressed at the hardware level, applying *software reengineering services* to energy efficiency mainly addresses applications in mobile computing. Thus, only energy efficiency in mobile computing will be considered in the following.

B. System Level

Another way of classifying past and current research is by looking at the level of a computer system they address.

Hardware. The most direct approach to energy efficiency optimization is building hardware that consumes less energy [7], or, put differently, wastes less energy by enabling better utilization.

Low-level Software. Software running on a given hardware can be optimized on a machine code level, e.g. by implementing such optimization into a compiler. Research on this level has been ongoing longer than on higher levels [3], [6], and as such has already been more comprehensively explored.

Operating Systems. The OS has control over hardware utilization. Given appropriate information and the required hardware capabilities, it can put certain components into sleep mode while they are not needed, or shut them down entirely. Scheduling may present further opportunities for energy savings [16]–[18].

Applications. The highest level are user-facing applications. Any energy optimizations on this level are dependent on the capabilities of the underlying hardware and operating system. On lower levels, application information cannot be taken into account, motivating the need for application-level power management [19], [20].

Software reengineering related consideration of energy efficiency on system level mainly refers to applications and their embedding in operating systems. Reengineering services are explored to analyze applications in two directions: (1) measure and improve the energy consumption by certain applications and (2) provide information on the behavior of an application to enable the operating system to ensure a more energy efficient scheduling of system processes serving the application.

III. Research Directions

As a result of our investigation into the state of the art of energy-efficient applications, both in industry and current research, we identify the following topics to be considered for further research: *A. Energy consumption and performance*, *B. Influencing energy consumption*, *C. Energy-consuming components*, *D. Scheduling in mobile computing*, *E. Application usage analysis for energy-awareness*.

The following sections summarize open research questions and domains exhibiting potential to save energy, and therefore offer research opportunities. As motivated in the previous section, the main focus is set on the domain of *mobile computing* and energy-saving techniques applied at the *application level*.

The topics presented here are revisited in Section V, giving examples of how they can be approached with software reengineering services.

A. Energy consumption and performance

Performance, in the sense of execution time, is strongly related to energy consumption. More powerful devices usually have higher energy demands, e.g. when comparing a notebook computer with a mobile phone (which is less powerful, but usually has a considerably higher average battery life) [21]. Modern processors often provide different power states, allowing to save energy at the cost of performance [22]. Another option is to completely switch off components not required by currently running applications. In desktop computers, the standardized *Advanced Configuration and Power Interface* (ACPI) gives operating systems control over power management. However, the operating system cannot make well-informed decisions to preserve energy if it is unaware of specific needs of running applications. For this, power management would have to move to the application level [19], [23].

Conversely, optimizing utilization of a given hardware platform will often both speed up execution, and lower energy consumption [3]. This can be achieved, e.g. by parallelizing instructions each using different processor parts. Leaving parts of the hardware idle usually wastes energy. So far, most optimizations have focused on hardware or instruction level optimizations, though the work done on application-level, e.g. Chung et al. [24], suggests opportunities for significant energy optimizations exist.

This leads to two research goals: first, empowering applications to pass on energy-related information to the operating system for smarter power management. And, second, identifying opportunities in software for better hardware utilization.

B. *Influencing energy consumption*

To improve a system's energy consumption, opportunities to save energy have to be identified. The previous section already revealed two such opportunities. Following, we list further areas which have been shown to present untapped potential for further energy optimizations:

Balasubramanian et al. [25] have developed a protocol reducing the energy cost associated with 3G, GSM, and WiFi data transmission, making use of knowledge about the requirements of the applications requesting data. In Atasu et al. [26], a technique for automatically designing instruction-set extensions for reduced energy consumption is described, utilizing the processors special capabilities for such adaptation. Amur et al. [27] have presented an approach to monitor running applications and predict idle times. This is leveraged by setting the CPU into an energy-conserving state while idle. Barr and Asanovic [28] have shown that wireless transmission of compressed data often uses more energy than transmitting uncompressed data, due to the energy cost of compressing and uncompressing. However, choosing an energy-optimal strategy has to take context information into account.

Further research should concentrate on exploiting such optimization potentials on the one hand, and on the other hand identify more areas where energy can be saved, by measuring and analyzing systems.

C. *Energy-consuming components*

A mobile phone can be broken down into several components, which all require electric energy to operate. Among them, for example, are transceivers connecting the phone to cellular networks, displays, cameras, CPUs and memory. Smartphones usually offer additional connectivity through WiFi, Bluetooth, and GPS, all requiring additional hardware components providing these capabilities.

Applications running on a device utilize these components to a different extent. Some parts of the system may not be required to be activated at all during certain times, or could at least be put into an energy-saving state. For example, monitoring user activity and utilizing location-awareness, user profiles for being *at home* and *at work* could be derived. If an accessible WiFi network was only available at home, the phone's WiFi component could be powered down while at work.

To utilize resources more energy-efficiently, first the energy consumption of single components has to be established, along with the degree of control the operating system or running applications have over their state. Then, applications would need to make information available to power management, regarding required components. Some recent studies have taken steps in this direction [25], [29].

D. *Scheduling in mobile computing*

Another potential area for energy optimization is the operating system's process scheduling system. Weiser et al. [16] have presented scheduling techniques trying to predict idle times to clock down the CPU. This approach has become a common feature of power management implementations as *dynamic voltage and frequency scaling*.

With this ability, quality of service of an application can be degraded in favor of lower energy consumption, by scaling down the processor and slowing the applications' execution. Often, though, it is more energy-efficient to execute a task as fast as possible to stay idle longer [30].

Specifically aimed at the smartphone domain, Calder and Marina [17] propose scheduling recurrent jobs in batches to reduce the amount of times the phone has to be woken up from an energy-saving sleep state.

With additional knowledge about the running applications, their current and future performance requirements, further optimization can be achieved. An example for such an approach is described by Yuan and Nahrstedt [18], whose scheduler optimizes for multimedia-related processes and as such makes assumptions about their *soft real-time* requirements.

E. *Application usage analysis for energy-awareness*

As discussed in Section III-B, to influence and optimize energy consumption, knowledge of saving potentials is required. This leads to the questions of what data can and should be collected, on which system level. One important dimension is the energy consumption of different system components, as described in Section III-B. This can be used to establish an energy model [25], [29] of the hardware platform used.

On the application level, information regarding the components actually needed by running applications is required, to optimize management of the hardware. It might also be worthwhile to profile long-term application, system, or user behavior, to be able to dynamically adapt power management to emerging usage patterns, and anticipate their occurrence. For example, Harris and Cahill [31] employed bayesian networks to combine different environment information sources and predict when a desktop computer could be powered down. Shye et al. [32] have monitored user activities on a smartphone and used the information to extract usage scenarios in which display brightness or CPU frequency could be reduced.

With both information – the current energy-state of the system, and the usage context (other running applications and the services they demand) – applications could make purposeful decisions to conserve energy.

The technical infrastructure to collect information about a system's energy consumption is available on many systems: hardware components providing information about their energy usage, ACPI or similar APIs on the OS-level, and application-level tools like *PowerTOP* [33] exist.

IV. Software Reengineering Services

Software reengineering is the modification of an existing software system to improve the system's quality.

Software reengineering is most often done in the context of the evolution of legacy software systems, e.g. to improve maintainability, or in preparation of subsequent migration efforts. The general reengineering process is summarized in the *horseshoe model* [34]. It consists of reverse engineering to extract a higher-level model of the system under study. On this level, the system is then *analyzed* and *restructured* with respect to reengineering goals and analysis results. Eventually the original abstraction level is reached by forward engineering, e.g. generating source code from the improved high-level model.

A number of services are used in software reverse and reengineering to gain the required knowledge about the system at hand (*program analysis*) and make systematic changes to enhance it in the desired way (*restructuring*). A short overview over each area is presented in Sections IV-A and IV-B, respectively. Following that, the research directions pointed out in Section III are revisited in Section V, to illustrate how each topic can be approached from a software reengineering perspective.

A. Program Analysis

Program analysis can be categorized as either *static* or *dynamic* [35]. Static analysis looks at the structure and composition of software systems, without taking into account run-time behavior, i.e. the system's inputs and resulting state. Two examples for common static analysis activities are:

- *Code smell detection* [36], the identification of patterns known or suspected to be detrimental to software quality (especially maintainability). In the same way, energy-wasting code patterns can be defined. This is tightly linked to *refactoring* (Sec. IV-B).
- *Software metrics* [37] are used to quantify certain system properties, e.g. energy indexes based on code patterns with known energy cost.

Dynamic analysis studies running systems [38]. Activities of dynamic program analysis include:

- *Source code instrumentation*, used to record execution traces of running applications. Together with recorded energy consumption, activities corresponding to high energy usage can be identified. Instrumentation can, for example, be achieved with frameworks supporting aspect-orientation.
- *Dynamic metrics*, e.g. how often a method is invoked. Energy optimization can focus on such *hot-spots*, representing code parts most often executed.

B. Restructuring

Restructuring a software system is *perfective maintenance*, i.e. it entails modifications which preserve functionality [4]. The abstraction level is also kept, excluding forward and reverse engineering steps.

In software reengineering, restructurings are applied to remove deficiencies from a software system identified by analysis, improving its quality. In the context of object orientation, the term *refactoring*, introduced by Fowler [36], is sometimes used synonymously with restructuring. It is, however, associated with Fowler's catalog of refactorings, a collection of (anti-) patterns and code smells, each encapsulating commonly found source code constructs, which are considered bad design.

V. Reengineering towards Energy-Efficient Applications

Software reengineering aims at improving software quality. Though usually thought of in software maintenance contexts (improving quality attributes like maintainability), the same techniques are useful to drive software systems towards energy efficiency. The following sections revisit research directions presented in Sec. III, and give examples of how software reengineering can be applied to further energy efficiency of (mobile) applications.

A. Energy consumption and performance

Two challenges have been presented concerning *energy consumption and performance*: *power management* on the application level, and *optimizing source code* for better hardware utilization.

Power management is actually present in all areas presented here, as energy optimization is approached mainly on the application level. The general idea is to use static and dynamic program analysis to gather information about applications' energy needs and expected behavior. More concrete examples are given in the following sections.

Optimizing source code for better energy efficiency can be approached with static analysis akin to the detection of code smells. Code patterns which could be optimized for energy efficiency can be derived from the target hardware architectures, and some examples have already been presented in literature: Cattoor et al. [39] propose restructuring certain loop constructs in a way that minimizes memory accesses. The approach by Chung et al. [24] depends on run-time information, to optimize code for situations which occur very often during its execution. Such information can be obtained by dynamic analysis.

To identify further patterns of energy-wasting source code, the energy consumption of a system executing different algorithm alternatives can be measured and analyzed (this idea is picked up again in Section III-B).

A catalog of energy-inefficient source code patterns could also be used to derive software metrics to measure and quantify energy efficiency of code, and classify applications according to such an energy rating.

B. Influencing energy consumption

The following examples for ways to influence energy consumption were given in Section III-B: wireless data transmissions, utilizing special processor capabilities, managing and exploiting processor idle times to go into energy-saving states, and trading off data compression and decompression costs against data transfer cost.

In general, knowledge about running applications can help make more accurate assumptions and predictions regarding the system's state and dynamics in the imminent future. For example, Amur et al. [27] predict running applications' activation and idle times to avoid scheduling timing interrupts when the processor is idle and sleeping. These information could be gathered by monitoring running applications using dynamic analysis services.

Another possible scenario may exploit the fact that, depending on certain circumstances, it may be either more energetically efficient to compress and decompress data, or sending it uncompressed [28]. Refactorings could be developed to introduce *strategy patterns* [40] into source code, allowing an application to dynamically decide whether to use compression or not. This may be based on current system state and the required performance or quality of service.

C. Energy-consuming components

Knowing which usage scenarios and components contribute to total energy consumption is an important prerequisite to systematically optimize for lowest possible energy consumption. The resulting energy model will enable applications and the operating system to make energy-aware decisions.

Gathering this knowledge can be done by attaching appropriate measurement instruments to a device, and its components [29]. This is, of course, an invasive procedure, which is not always possible.

Measurement can also be approached in software [25], given that both hardware components and the operating system offer appropriate capabilities to attain energy consumption information. With this approach, an energy model can be established by an application at run-time. This can be coupled by dynamic analysis, e.g. recording execution traces to match usage scenarios to energy usage. It may also provide enough information to deduce the energy consumption of single components from the overall consumption, in case the device does not offer fine-grained energy-related information.

D. Scheduling in mobile computing

Section III-D illustrated that the CPU scheduler is a central component to optimize processor utilization for energy efficiency. To do so, it requires information about running applications, regarding, e.g. latency criticality or periodic wake-ups.

Reengineering services can be used to collect these data, e.g. by monitoring running applications using code instrumentation. The resulting information is further analyzed for patterns, both of a single application, and between different applications running at the same time.

Applications with similar activation and idle patterns can then be scheduled in batches [17]. Another possible optimization opportunity would be to schedule applications, which largely utilize different parts of the CPU, at the same time and try to parallelize instructions.

The ability to influence the scheduling of processes from the application level is, however, limited in most operating systems. In this regard, the viability of energy-saving approaches aimed at scheduling, remains an open question.

E. Application usage analysis for energy-awareness

Application usage analysis aims at monitoring user activity and exploiting detected usage patterns for energy optimization. This requires combining different reengineering services: code instrumentation can be used to monitor running applications. This information has to be analyzed in context: all running applications have to be taken into account, as optimization efforts may otherwise cancel each other out. System constraints like battery charge also have to be considered. Further dynamic analysis has to take all these information sources, and map application activity, component usage and energy consumption.

This motivates the need for a central power manager on application level, ideally based on a standardized, platform-independent interface providing information relevant to energy optimization. The design of such an *energy abstraction layer* is central to the research agenda, presented in the following section.

VI. Research Agenda

In this section an overview over planned next steps to conduct future research into energy efficient applications using *software reengineering services* is presented.

First, a suitable infrastructure has to be built. This includes making energy consumption measurable, i.e. define measuring means to collect energy-related data, utilizing the capabilities available on different platforms and operating systems, and encapsulating them under a unified interface. This interface will also include services to support the operating system with application-specific informations to optimize energy consumption of the entire system. As such, it constitutes an *energy abstraction layer* between the operating system, analysis tools, and energy-efficient applications.

Also required are the means to analyze applications and, by extension, user activities. To this end, application code has to be instrumented, for example using aspect-orientation frameworks. With the measurement infrastructure in place, further analyses are required to identify mappings between energy consumption of the system, components, running applications, and user behavior. Having made these connections, this data will be analyzed for patterns to make predictions about system activity, and optimize accordingly.

Parallel to these activities mainly supported by static and dynamic analysis services, refactoring for energy efficiency will be researched. Energy-inefficient code patterns have to be identified, either from previous research work into this topic, or empirically by measurement and analysis. Patterns can then be detected in source code like *bad smells* through static analysis, and be used to develop suitable refactorings to replace them with functionally equivalent, but more energy-efficient alternatives.

While this paper focused on technical aspects, issues of how to incorporate energy saving techniques into real-world development practices remain as open question. In particular, the viability of additional development effort to create more energy efficient applications, both economically and ecologically, has to be considered. Another interesting concern for further research would be how refactoring towards energy efficient code affects maintainability.

VII. Summary

This paper gave an overview on current research into energy efficiency in information and communication technology. Further improvements of providing energy efficiency on an application level were motivated by applying reengineering services.

References

[1] L. Stobbe, N. Nissen, M. Proske, A. Middendorf, B. Schlomann, M. Friedewald, P. Georgieff, and T. Leimbach, "Abschätzung des Energiebedarfs der weiteren Entwicklung der Informationsgesellschaft," Fraunhofer ISI and IZM, Berlin, Abschlussbericht an das Bundesministerium für Wirtschaft und Technologie, 2009.

[2] W. Nebel, M. Hoyer, K. Schröder, and D. Schlitt, "Untersuchung des Potentials von rechenzentrenübergreifendem Lastmanagement zur Reduzierung des Energieverbrauchs in der IKT," OFFIS, Oldenburg, Studie für das Bundesministerium für Wirtschaft und Technologie, 2009.

[3] K. Roy and M. C. Johnson, "Software design for low power," in *Low power design in deep submicron electronics*, W. Nebel and J. P. Mermet, Eds. Berlin: Springer, 1997, pp. 433–460.

[4] E. J. Chikofsky and J. H. Cross, II, "Reverse engineering and design recovery: A taxonomy," *IEEE software*, vol. 7, no. 1, pp. 13–17, 1990.

[5] K. Naik, "A survey of software based energy saving methodologies for handheld wireless communication devices," University of Waterloo, Tech. Rep., 2010.

[6] S. Devadas and S. Malik, "A survey of optimization techniques targeting low power VLSI circuits," in *Proceedings of the 32nd annual ACM/IEEE Design Automation Conference*. ACM, 1995, pp. 242–247.

[7] A. Chandrakasan and R. Brodersen, *Low power digital CMOS design*. Springer, 1995.

[8] R. Min, M. Bhardwaj, S. Cho, E. Shih, A. Sinha, A. Wang, and A. Chandrakasan, "Low-power wireless sensor networks," in *VLSI Design, 2001. Fourteenth International Conference on*. IEEE, 2001, pp. 205–210.

[9] V. Raghunathan, C. Schurgers, and M. Srivastava, "Energy-aware wireless microsensor networks," *IEEE Signal Processing Magazine*, vol. 19, no. 2, pp. 40–50, 2002.

[10] G. Anastasi, M. Conti, M. Di Francesco, and A. Passarella, "Energy conservation in wireless sensor networks: A survey," *Ad Hoc Networks*, vol. 7, no. 3, pp. 537–568, 2009.

[11] C. Jones, K. Sivalingam, and P. Agrawal, "A survey of energy efficient network protocols for wireless networks," *wireless networks*, pp. 343–358, 2001.

[12] M. Poess and R. O. Nambiar, "Energy cost, the key challenge of today's data centers: a power consumption analysis of TPC-C results," *Proceedings of the VLDB Endowment*, vol. 1, no. 2, pp. 1229–1240, 2008.

[13] A. Berl, E. Gelenbe, M. Di Girolamo, G. Giuliani, H. De Meer, M. Q. Dang, and K. Pentikousis, "Energy-Efficient Cloud Computing," *The Computer Journal*, vol. 53, no. 7, pp. 1045–1051, 2009.

[14] A. J. Younge, G. von Laszewski, L. Wang, S. Lopez-Alarcon, and W. Carithers, "Efficient resource management for Cloud computing environments," in *International Conference on Green Computing*. IEEE, 2010, pp. 357–364.

[15] K. Schröder, D. Schlitt, M. Hoyer, and W. Nebel, "Power and cost aware distributed load management," in *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking - e-Energy '10*. New York, USA: ACM, 2010, p. 123.

[16] M. Weiser, B. Welch, A. Demers, and S. Shenker, "Scheduling for reduced CPU energy," *Mobile Computing*, pp. 449–471, November 1996.

[17] M. Calder and M. Marina, "Batch Scheduling of Recurrent Applications for Energy Savings on Mobile Phones," *Sensor Mesh and Ad Hoc Communications and Networks (SECON), 2010 7th Annual IEEE Communications Society Conference on*, pp. 1–3, 2010.

[18] W. Yuan and K. Nahrstedt, "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 149–163, 2003.

[19] C. C. Ellis, "The case for higher-level power management," in *Proceedings of the Seventh Workshop on Hot Topics in Operating Systems*. IEEE CS, 1999, pp. 162–167.

[20] X. Liu and P. Shenoy, "Chameleon: Application-level power management," *Mobile Computing, IEEE*, vol. 7, no. 8, pp. 995–1010, 2008.

[21] T. D. Burd and R. W. Brodersen, *Energy Efficient Microprocessor Design*. Kluwer Academic Publishers, 2002.

[22] E. Saxe, "Power-Efficient Software," *Queue*, vol. 8, no. 1, p. 10, 2010.

[23] F. Shearer, *Power management in mobile devices*. Newnes, 2007.

[24] E.-y. Chung, L. Benini, and G. D. Micheli, "Energy Efficient Source Code Transformation based on Value Profiling," *Transformation*, 2000.

[25] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A Measurement Study and Implications for Network Applications," in *Proceedings of the 9th ACM SIGCOMM Internet measurement conference*. ACM, 2009, pp. 280–293.

[26] K. Atasu, L. Pozzi, and P. Ienne, "Automatic Application-Specific Instruction-Set Extensions under Microarchitectural Constraints," *International Journal of Parallel Programming*, vol. 31, no. 6, pp. 411–428, 2003.

[27] H. Amur, R. Nathuji, M. Ghosh, K. Schwan, and H.-H. S. Lee, "IdlePower: Application-Aware Management of Processor Idle States," in *MMCS*, 2008.

[28] K. Barr and K. Asanovic, "Energy Aware Lossless Data Compression," in *MobiSys2003*, 2003, pp. 231–244.

[29] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *USENIXATC'10 Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, 2010, p. 21.

[30] M. Garrett, "Powering down," *Communications of the ACM*, vol. 51, no. 9, pp. 42–46, 2008.

[31] C. Harris and V. Cahill, "Exploiting user behaviour for context-aware power management," in *WiMob'2005, IEEE International Conference on Wireless And Mobile Computing, Networking And Communications, 2005.*, vol. 4. IEEE, 2005, pp. 122–130.

[32] A. Shye, B. Scholbrock, and G. Memik, "Into the wild," in *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42*. New York, USA: ACM Press, 2009, p. 168.

[33] Intel Corporation, "PowerTOP," 2007. [Online]. Available: http://www.linuxpowertop.org/

[34] R. Kazman, S. G. Woods, and S. J. Carriere, "Requirements for Integrating Software Architecture and Reengineering Models: CORUM II," in *Proceedings of the Working Conference on Reverse Engineering (WCRE'98)*, 1998, p. 154.

[35] D. Binkley, "Source code analysis: A road map," in *2007 Future of Software Engineering*. IEEE CS, 2007, pp. 104–119.

[36] M. Fowler and K. Beck, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[37] N. E. Fenton, *Software Metrics: A Rigorous Approach*. Chapman and Hall, 1991.

[38] B. Cornelissen, A. Zaidman, A. van Deursen, L. Moonen, and R. Koschke, "A Systematic Survey of Program Comprehension through Dynamic Analysis," *Software Engineering, IEEE Transactions on*, vol. 35 (5), pp. 684–702, 2009.

[39] F. Catthoor, F. Franssen, S. Wuytack, L. Nachtergaele, and H. De Man, "Global communication and memory optimizing transformations for low power systems," in *IEEE International Workshop on Low Power Design*. IEEE, 1994, pp. 203–208.

[40] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.