ENERGY AWARENESS NEEDS A RETHINKING IN SOFTWARE DEVELOPMENT

Hagen Höpfner[1] and Christian Bunse[2]
[1]*Mobile Media Group, Bauhaus-University of Weimar, Bauhausstr. 11, 99423 Weimar, Germany*
[2]*Software Systems, University of Applied Sciences of Stralsund, Stralsund, Germany*

Keywords: Software engineering, Data processing, Energy awareness.

Abstract: Energy efficiency and -awareness are buzzwords in various areas of information and communication technology and are at the core backbone of GreenIT. Computing centers aim for reducing energy consumption in order to save money and carbon dioxide emissions. Furthermore, GreenIT labels are perfect selling points for computer equipment. Especially battey-powered and mobile devices must consider software's energy consumption in order to prolong their uptime, while keeping the desired or agreed quality of service (QoS). Even if energy awareness regarding hardware has been researched intensively for a couple of years, the analysis of the impact of software on energy consumption is rather novel. We claim that the development of energy-efficient and -aware software systems require a careful re-examination of the many paradigms in software development.

1 MOTIVATION

The history of software engineering shows that innovation is often stipulated by antedecent improvements in the hardware sector. In the "good old days" developers had to understand registers, CPU-opcodes, or interrupt handling in order to develop professional software systems. They had to optimize source- and assembly code in order to improve performance and quality. Interestingly, this is still true in some domains (e.g., automotive) where resources are scarce. In the following years, software became more and more complex and development paradigms like object-orientation, or model-driven and component-oriented software engineering allowed for distinct and clearly separated views regarding hardware and software *kiwi*-optimization. *kiwi* is an acronym for "kill it with iron" and describes the approach to solve performance problems with additional resources or hardware. Unfortunately, *kiwi* does not solve every optimization problem regarding the non-functional properties of a software system. Reliability, safety or security may need substantial adaptations of the system and not just new, additional resources.

A a rule of thumbs it that more complex software requires more energy. However, functional requirements demand a certain complexity. So, making software less complex is not a proper approach for reducing energy consumption. Developers and researchers must rethink many paradigms in software development if they want to reduce software dependent energy consumption. In our research we addressed various hypotheses on how to develop energy-efficient software. This position paper will briefly introduce our ideas and support our opinion that energy awareness requires a rethinking in software development.

2 ALGORITHMS

For the last couple of years, we taught our students that various implementation alternatives for certain problems exist and that algorithms that need less CPU-cycles are usually the better choice. Complexity theory, which aims on describing the boundaries of the number of required CPU-cycles or operation steps as a function over the number of input data, formed the basis for the performance prediction of an implementation alternative. If we look, e.g., on different sorting algorithms, we prefer Mergesort to Insertionsort because the complexity of Mergesort is $O(n \cdot \log(n))$ while the complexity of Insertionsort is $O(n^2)$. On the one hand, both algorithms can be used for sorting an input list with n elements and Mergesort needs less comparison steps than Insertionsort. On the other hand, Mergesort is a recursive algorithm that reduces the number of comparisons through a di-

Höpfner H. and Bunse C..
ENERGY AWARENESS NEEDS A RETHINKING IN SOFTWARE DEVELOPMENT.
DOI: 10.5220/0003578002940297
In *Proceedings of the 6th International Conference on Software and Database Technologies* (ICSOFT-2011), pages 294-297
ISBN: 978-989-8425-77-5

vide and conquer strategy. Hence, it requires more memory than Insertionsort. In previous works we analyzed the energy consumption of different sorting (Bunse et al., 2009a) and join algorithms (Höpfner and Bunse, 2010a) and showed that there is *no* statistical correlation between energy consumption and computational complexity Even if Insertionsort takes more time to finish its task, it consumes less energy.

Further research (cf. (Bunse et al., 2009b)) has shown that energy consumption can be reduced by switching between different algorithms based on their energy efficient operational range and on the input data size. However, if it comes to energy-aware software, algorithm designers and programmers must formally understand the energy consumption of an algorithm: The simple assumption that faster algorithms require less energy as they finish faster is wrong! Thus, we have to find a complexity measure that, similar to computation complexity, allows us to specify boundaries for an algorithm's energy consumption. We have to accept, that especially for battery-powered devices, processing a limited amount of data, slower but energy-efficient algorithms are preferable to faster but energy intensive alternatives.

3 RESOURCE SUBSTITUTION

Almost all computers, smartphones, laptops, embedded systems, etc. use a Von-Neumann-Architecture (von Neumann, 1993): Data and program code are stored in main memory and a CPU accesses them over a bus system. Ubiquitous availability of network accesses enables information exchange and task distribution among devices with different built-in resources. They form clouds that are either used to store data or to distribute workload. Thus, various hardware resources with different energy consumptions are utilized for data processing and storage. However, certain resources can be substituted by other ones (cf. (Höpfner and Bunse, 2007)). Several projects analyzed such resource substitution strategies with regard to energy consumption. The authors of (Marwedel, 2007) state that CPU usage needs comparatively less energy than memory storage. The authors of (Veijalainen et al., 2004) state that compressing a file by more then 10%, transmitting and decompressing it requires less energy than transmitting it uncompressed. According to the authors of (Kansal and Zhao, 2008), file compression also reduces the energy consumption of operations on hard disks.

Substituting resources only reflected/reflects the capacity of the involved resources. Compression was/is used to save storage memory or communication afford. View materialization in database management systems reduces the number of executions of CPU-intensive view definition queries by storing/materializing the queries results. Task distribution like the well known SETI@home helped/helps to realize computations that require additional CPU-cycles. However, only a few researchers tried to rethink the idea of resource substitution while focussing on energy awareness. In (Höpfner and Bunse, 2010b) we started to summarize these works and started to define a classification scheme for energy related aspects of a resource substitution strategy.

4 INFORMATION QUALITY

The previousely discussed issues focussed on energy aspects while maintaining the quality of processed information. However, certain application scenarios (e.g. mobile information systems (Veijalainen and Gross, 2002)) allow for trading information quality off energy consumptions (i.e., graceful degradation). A similar idea has been introduced in terms of lossy compression for reducing the size of multimedia files: One trades the quality of an image, a movie, or an audio file off its file size. Especially in networked environments, this approach is common as users accept lower quality to a certain extent. With regard to energy consumption, one has to adapt this idea. Of course, transmitting smaller files needs less energy than transmitting larger ones, but this issue was already addressed. In addition, energy-aware lossy compression algorithms might help to find a balance between information quality (quality of a video stream, etc.) and energy consumption.

The authors of (Hüls, 2002) found out that the energy consumption of an MPEG-algorithm is largely affected by the used data types. By replacing `double` by `float`, the algorithm's energy consumption was reduced to 34%. Of course, using less precise numbers leads to a drastic reduction of the resulting quality of the MPEG-file. Our own experiments on sorting algorithms (Höpfner and Bunse, 2010a) show that although data sizes were simply doubled (2-Byte integer was replaced by 4-Byte floating point numbers), the algorithm requires significantly more energy. Interestingly, the differences are not simply correlated with a factor of two. Beneath additional memory requirements, one reason for this growth might also be the software emulated realization of the floating point unit of the processor used for this experiment.

The conclusion for this section is twofold. On the one hand, energy efficiency strongly depends on the correct choice of data types. This issue is al-

ready reflected per default in software engineering and database design, as proper data types should be chosen anyway. On the other hand, compromises on the information quality allow to use less complex data types and therefore to reduce energy consumption. However, we are not aware of any systematic research on the user acceptance in such a scenario.

5 SOFTWARE ENGINEERING

Energy awareness and energy optimization strategies have to be integrated into the processes and methods of software development. This requires means for specifying energy related properties as well as prediction and optimization approaches.

A literature review has shown that quantitative descriptions of system dependability are generally done via combinations of attributes: Reliability, availability, safety, integrity, confidentiality and maintainability. In modern component based software development (CBSD), these attributes are validated by modeling the system components using binary fault states and by combining the fault states to an overall system state using Boolean logical function. The fault states themselves are usually modeled as probabilistic faults due to component wear out or other probabilistic sources. In practice, a variety of techniques, languages and tools are used for modeling both hardware and software. Examples are SysML, Modelica, Matlab, FMEA/FMECA, RBD, FTA, Markov chains, and Petri-nets. However, these techniques do not address the challenges of the GreenIT-era, nor do they always follow sound engineering principles.

Specification and modeling of software components is a much less mature discipline. At code level, there are three well-known component models: Microsofts .NET, Suns Enterprise Java Beans and the OMG's CORBA Component Model, each with its own interface definition language (IDL). However, IDL specifications are purely syntactic and do not contain a semantic description of a components functionality. At "model" level, the UML2.0 component model offers a rich specification of components in terms of various UML diagrams, but much of the semantic information is missing. The semantically richest specification languages are currently associated with web services and service-oriented architectures. The Web Service Description Language provides syntactic information like the component IDLs. However, a new suite of semantic service specification languages such a WSDLS, OWLS and METEORS have recently been developed to address this problem. None of the aforementioned technologies provides a development process, and although there are several methods, that aim to support specific aspects of CBSD (such as component identification), no one is integrated in mainstream development processes.

For many years, non-functional requirements were written textually as part of the overall system requirements documents. However, with the increasing use of hardware and software components and the commoditization of computing power via grids, the specification of the QoS requirements has gained increasing attention. At the hardware level, QoS has long been specified in terms of bandwidth and fault probability. At the software level, several languages have recently been introduced for the specific purpose of modeling and describing QoS requirements. Moreover, the web service technology also includes mechanisms specifically for defining QoS requirements and policies such as WSPolicy. To date, however, there are no mainstream methodologies focusing on the creation and use of energy-related QoS specifications. Interestingly, research on specifying the resources of software systems (Weilkiens, 2008; Paredis et al., 2010), (Graf et al., 2006; Thomas et al., 2008) have their focus on of platform models and performance. Energy consumption is widely neglected.

To overcome these problems and to also address the problem of optimization, another pillar of engineering energy-aware software systems, we developed MARMOT (Bunse et al., 2007), a method that facilitates reuse in embedded software system development. MARMOT is a component-based and model-driven development framework. Composition is a key activity in component-based development with MARMOT. Within MARMOT, the energy consumption of a software system can either be statically or dynamically, and one can optimize it at development or at runtime (Bunse and Höpfner, 2008).

A final aspect that is currently beyond the scope of our research is the prediction of the effects of changes or more complex actions such as software composition. Regarding quality properties such as reliability or maintainability, there exist already tools (cf. (Becker et al., 2007; Happe et al., 2010a; Happe et al., 2010b)). Regarding energy first projects (Kounev, 2011; Rathfelder et al., 2010) have been started but did not provide any results yet. In addition to prediction, engineering also requires support for optimization. Optimization can either take place at development or at runtime. At development time energy consumption can be improved by adapting system models, implementations, and infrastructures (Siegmund et al., 2010). At runtime a system might adapt itself (Grassi et al., 2009) according to external needs and requirements, such as energy consumption.

6 SUMMARY AND OUTLOOK

We claimed that developers and researchers have to rethink many paradigms in software development if they want to reduce software-dependent energy consumption. We illustrated that, so far, almost all well accepted-approaches starting from complexity theory issues, over resource substitution issues and information quality compromises, to the software engineering process do not properly support energy awareness. We presented some of our findings as well as research conducted by other researchers. However, all presented results need more generalization in order to form the basis and a profound framework for energy aware software development. The authors of this paper welcome all interested researchers to start the discussion about these issues.

REFERENCES

Becker, S., Koziolek, H., and Reussner, R. (2007). Model-Based performance prediction with the palladio component model. In *WOSP'07*, pages 54–65, New York, NY, USA. ACM.

Bunse, C., Groß, H.-G., and Peper, C. (2007). Applying a Model-based Approach for Embedded System Development. In *EUROMICRO'07*, pages 121–128, Los Alamitos, CA, USA. IEEE.

Bunse, C. and Höpfner, H. (2008). Resource substitution with components — optimizing energy consumption. In *ICSOFT'08*, volume SE/GSDCA/MUSE, pages 28–35, Setúbal, Portugal. INSTICC.

Bunse, C., Höpfner, H., Mansour, E., and Roychoudhury, S. (2009a). Exploring the Energy Consumption of Data Sorting Algorithms in Embedded and Mobile Environments. In *MDM'09*, pages 600–607, Los Alamitos, CA, USA. IEEE.

Bunse, C., Höpfner, H., Roychoudhury, S., and Mansour, E. (2009b). Choosing the "best" Sorting Algorithm for Optimal Energy Consumption. In *ICSOFT'09*, volume 2, pages 199–206. INSTICC.

Graf, S., Gérard, S., Haugen, Ø., Ober, I., and Selic, B. (2006). Modeling and Analysis of Real-Time and Embedded Systems. In *Satellite Events at MoDELS'05*, volume 3844 of *LNCS*, pages 58–66, Berlin / Heidelberg. Springer.

Grassi, V., Mirandola, R., and Randazzo, E. (2009). Model-Driven Assessment of QoS-Aware Self-Adaptation. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *LNCS*, pages 201–222, Berlin / Heidelberg. Springer.

Happe, J., Becker, S., Rathfelder, C., Friedrich, H., and Reussner, R. H. (2010a). Parametric Performance Completions for Model-Driven Performance Prediction. *Performance Evaluation*, 67(8):694–716.

Happe, J., Groenda, H., Hauck, M., and Reussner, R. H. (2010b). A Prediction Model for Software Performance in Symmetric Multiprocessing Environments. In *QEST'07*, pages 59–68, Los Alamitos, CA, USA. IEEE.

Höpfner, H. and Bunse, C. (2007). Ressource Substitution for the Realization of Mobile Information Systems. In *ICSOFT'07*, volume SE, pages 283–289, Setúbal, Portugal. INSTICC.

Höpfner, H. and Bunse, C. (2010a). Energy Aware Data Management on AVR Micro Controller Based Systems. *ACM SIGSOFT SE Notes*, 35(3).

Höpfner, H. and Bunse, C. (2010b). Towards an energy-consumption based complexity classification for resource substitution strategies. In *Proc. GVDB'10*, volume 581 of *CEUR Workshop Proc.* CEUR-WS.org.

Hüls, T. (2002). Optimizing the energy consumption of an MPEG application. Master's thesis, TU Dortmund, Fakultät für Informatik, Dortmund, Germany.

Kansal, A. and Zhao, F. (2008). Fine-grained energy profiling for power-aware application design. *SIGMETRICS Performance Evaluation Review*, 36(2):26–31.

Kounev, S. (2011). Self-Aware Software and Systems Engineering: A Vision and Research Roadmap. In *SE'11, Nachwuchswissenschaftler-Symposium*.

Marwedel, P. (2007). *Embedded System Design*. Springer.

Neumann, J. V. (1993). First Draft of a Report on the EDVAC. *IEEE Annals of the History of Computing*, 15(4):27–75.

Paredis, C. J., Bernard, Y., Burkhart, R. M., de Koning, H.-P., Friedenthal, S., Fritzson, P., Rouquette, N. F., and Schamai, W. (2010). An Overview of the SysML-Modelica Transformation Specification. In *INCOSE Int. Symposium 2010*.

Rathfelder, C., Klatt, B., Brosch, F., and Kounev, S. (2010). Performance Modeling for Quality of Service Prediction in Service-Oriented Systems. In *Handbook of Research on Non-Functional Properties for Service-Oriented Systems: Future Directions*. IGI Global.

Siegmund, N., Kuhlemann, M., Pukall, M., and Apel, S. (2010). Optimizing non-functional properties of software product lines by means of refactorings. In *VaMoS'10*, volume 37 of *ICB-Research Report*, pages 115–122. Uni Duisburg-Essen.

Thomas, F., Gérard, S., Delatour, J., and Terrier, F. (2008). Software Real-Time Resource Modeling. In *Embedded Systems Specification and Design Languages*, volume 10 of *LNEE*, pages 169–182. Springer, Berlin / Heidelberg.

Veijalainen, J. and Gross, T. (2002). Mobile Wireless Interfaces: In Search for the Limits. In *Developing an Infrastructure for Mobile and Wireless Systems*, volume 2538 of *LNCS*, pages 153–163, Berlin / Heidelberg. Springer.

Veijalainen, J., Ojanen, E., Haq, M. A., Vahteala, V.-P., and Matsumoto, M. (2004). Energy Consumption Tradeoffs for Compressed Wireless Data at a Mobile Terminal. *IEICE ToC*, E87-B(5):1123–1130.

Weilkiens, T. (2008). *Systems Engineering mit SysML/UML — Modellierung, Analyse, Design*. dPunkt.Verlag, Heidelberg, 2nd edition.