An Energy Consumption Perspective on Software Architecture

A Case Study on Architectural Change

Erik A. Jagroep[1,2], Jan Martijn E. M. van der Werf[1], Ruvar Spauwen[1], Leen Blom[2], Rob van Vliet[2], and Sjaak Brinkkemper[1]

[1] Department of Information and Computing Science
Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{e.a.jagroep,j.m.e.m.vanderwerf,r.a.spauwen,s.brinkkemper}@uu.nl,
[2] Centric Netherlands B.V.
P.O. Box 338, 2800 AH Gouda, The Netherlands
{leen.blom, rob.van.vliet}@centric.eu

Abstract. The rising energy consumption of the ICT industry has triggered a quest for more sustainable, i.e. energy efficient, ICT solutions. Software plays an essential role in finding these solutions, as software is identified as the true consumer of power. However, in this context, software is often treated as a single, complex entity which fails to provide detailed insight in the elements that invoke specific energy consumption behavior.
In this paper, we propose an energy consumption perspective on software architecture as a means to provide this insight and enable analysis on the architectural elements that are the actual drivers behind the energy consumption. In a case study using a commercial software product, the perspective is applied and its potential demonstrated by achieving an energy consumption saving of 67.1%.

Keywords: Software architecture · Energy consumption perspective · Sustainability

1 Introduction

The energy consumption of the Information and Communication Technology (ICT) sector is a booming topic of interest. Recent figures indicate that at least a tenth of the world's electricity use is on behalf of ICT [8]; a figure that has kept growing over the years. As a result of the increased awareness on the subject, the term 'sustainability' has emerged which is to "meet the needs of the present without compromising the ability of future generations to satisfy their own needs" [9]. Within the research community this has resulted in much attention going towards increasing the energy efficiency of ICT.

Only recently the role of green software is stressed in finding sustainable ICT solutions [7]. While energy is directly consumed by hardware, the operations are directed by software which is argued to be the true consumer of power [14]. In current research on the Energy Consumption (EC) of software (cf. [3,4]), the software is often treated as a single, complex entity (i.e. considered on application level) instead of the inter-related

elements it actually consists of. A breakdown into hardware components and 'units of work' is made, but this does not provide insight into which modules and functions invoke specific energy consuming behavior. Consequently, a stakeholder can not direct sustainability efforts to where they are needed.

We argue Software Architecture (SA) is able to fill this gap and in this paper we investigate how EC can be positioned within the scope of SA. An Architecture Description (AD) complemented with EC measurements, has the potential to help determine appropriate adjustments, identify where they should be applied and help to simplify the context by limiting the scope. Using a commercial software product, we construct an EC perspective on SA and validate the perspective through a case study. The potential of our research is demonstrated by realizing a reduction in energy consumption of 67.1%.

In this paper we first present related work on energy consumption and SA (Sect. 2). After this brief introduction we continue with constructing the perspective alongside a case study (Sect. 3). Finally, we provide a conclusion, discuss the results and directions for future research (Sect. 4).

2 Green Software and Software Architecture

Our approach to analyze the EC of software on architectural level is not unique. The node map presented in [3] for example, closely resembles what could be labeled as a deployment view which, after including EC figures, provides a 'heat map' of the system. Following this same line [4] presents the 'ME^3SA' model in which again the deployment and functional components of the software are investigated. In relation to green software, a limitation of both approaches is that most recommendations relate to hardware aspects and only provide 'strong clues' on software level.

One of the main issues with respect to green software [7] is to perform detailed EC measurements. Specialized environments, e.g. [4], enable detailed measurements but often lack the ability to expand to more complex environments (e.g. data center) where other approaches, e.g. 'E-Surgeon' [10], are required that have their own limitations. Consequently, the EC of software is often measured by relating the hardware EC to computational resource usage on behalf of the software [3].

To perform EC measurements we expand on the call for sustainability to become a Quality Attribute (QA) with *resource consumption*, *greenhouse gas emissions*, *social sustainability*, and *recycling* as subcharacteristics [7]. Continuing on the path of EC we focus specifically on resource consumption which, following the ISO 25010 standard, can be quantified using quality properties and quality measures. From literature [2–6] three potential quality properties can be identified; *Software utilization* (the degree to which hardware resource utilization on the account of software meets requirements), *Energy usage* (the degree to which the amount of energy used by the software meets requirements) and *Workload energy* (the degree to which the EC related to performing a specific task using software meets requirements). In Table 1 the properties are broken down into quality measures complemented with a definition and measurement function. Although further research is required, for now we assume that these quality properties cover the resource consumption subcharacteristic.

Green Architectural Tactics To address concerns for a software product on the level of the SA, tactics are applied. A tactic is a decision that influences the control of a QA [1] and is a design option that helps the architect in realizing a desired property for a system. In relation to EC, there is still work to be done to find a set of tactics that are able to satisfy EC concerns. Consequently, the presented tactics are by no means definitive and should be considered as a source of inspiration for green software efforts.

In [11] a catalog is presented consisting of three categories, including tactics, that address energy efficiency in the cloud. The energy monitoring category tactics are aimed at collecting power consumption information and estimating infrastructure and software component power consumption. Tactics in the self-adaptation category present possibilities for optimization during run-time. Finally, the cloud federation tactics are aimed at respectively finding and switching to the most energy efficient services to perform a task. Although the tactics are explained specifically in a cloud computing context, they could prove valuable for software in general.

Increase hardware utilization [3]; Ineffective use of hardware is a common source for energy inefficiency and is one of the triggers to consolidate the number of active servers. From an EC point of view less hardware reduces the idle energy consumption.

Table 1. Quality measures for to the resource consumption subcharacteristic.

Resource consumption	
Software utilization	
CPU Utilization (CPUU)	Measure of the CPU load related to running the software. *current CPU load − idle CPU load*
Memory Utilization (MU)	Measure of the memory usage related to running the software. $\frac{\textit{allocated memory}}{\textit{total memory}} \times 100\%$
Network Throughput (NT)	Measure of the network load related to running the software. *Packages, sent/received bytes per second*
Disk Throughput (DT)	Measure of the disk usage induced by running the software. *Disk I/O per second*
Energy usage	
Software Energy Consumption (SEC)	Measure for the total energy consumed by the software. *EC while operating − idle EC*
Unit Energy Consumption (UEC)	Measure for the energy consumed by a specific unit of the software. $(\frac{\textit{Unit CPUU}}{\textit{CPUU}} \times \frac{\textit{Unit MU}}{\textit{MU}} \times \frac{\textit{Unit NT}}{\textit{NT}} \times \frac{\textit{Unit DT}}{\textit{DT}}) \times \textit{SEC}$
Relative Unit Energy Consumption (RUEC)	Measure for the energy consumed by a specific unit compared to the entire software instance. $\frac{\textit{UEC}}{\textit{SEC}} \times 100\%$
Workload energy	
Task Energy Consumption (TEC)	Measure for the energy consumed when a task is performed. $\frac{\textit{SEC}}{\text{\# of tasks performed}}$
Unit Task Energy Consumption (UTEC)	Measure for the energy consumed when a task is performed by a specific unit of the software. $\frac{\textit{UEC}}{\text{\# of tasks performed}}$

Concurrency architecture variation [16]; In this specific case the Half Synchronous / Half Asynchronous and the Leader / Followers concurrency architectures are compared and a significant difference was found in the advantage of the first. However, further investigation is required to test the generalizability of this finding.

Increase modularity; In terms of database calls, software consisting of fewer modules could require less calls while significantly more data is transferred per call. When software consists of more modules, an increase in database calls could be observed with the potential that less data is transferred per call, i.e. the calls are more fitted to the process. Assuming that increased disk usage has a marginal impact on the EC figures, less CPU capacity is required for processing the call thereby lowering the EC per call.

Network load optimization; Although modularity can positively affect the EC of software [15], more modules also implies a higher communication load. A positive effect on the EC is expected with a reduced communication load.

3 Energy Consumption Perspective on Software Architecture

To address the EC of software on an architectural level we propose to construct an EC perspective, which is 'a collection of activities, tactics, and guidelines ... used to ensure that a system exhibits a particular set of related quality properties that require consideration across a number of the systems architectural views' [12]. In order to create a comprehensive EC perspective, the perspective catalog [12] (Fig. 1) is used where for each viewpoint a key issue is formulated that addresses the relation to EC and a suggestion is provided on how the AD can be altered to adhere to the perspective. Note that the original catalog contains six viewpoints, but a seventh viewpoint, the 'context view', was added to define societal and economical aspects.

To increase the practical applicability of the perspective, it was created alongside a case study using Document Generator (DG). DG is a commercial software product, used as a service with other commercial software products, to generate over 30 million documents per year by over 300 customers. The case study was performed in a test environment (Fig. 2) that allowed for EC measurements using a WattsUp? Pro (WUP),

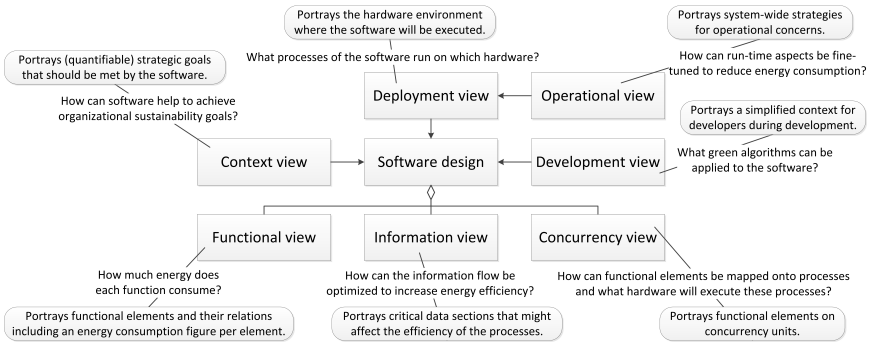

Fig. 1. Viewpoint catalog to apply an EC perspective, after [12], including key issues and AD alteration suggestions.

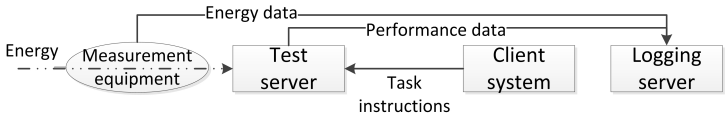

Fig. 2. Setup of the test environment used to perform the case study.

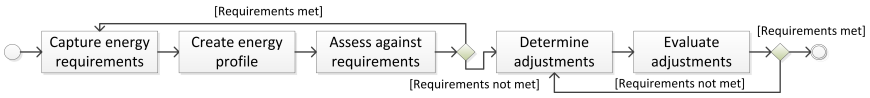

Fig. 3. The activities to apply the EC perspective to software architecture.

a device capable of measuring the total power drawn by an entire system with a one second interval between measurements, and performance measurements using Perfmon, a standard performance monitoring tool with Microsoft Windows. As DG was installed on the test server[3] measurements were performed on this system and data was collected with the loggin server. Finally, the client system was used to perform a task with DG.

Perspective Activities Using a perspective a stakeholder has a means to analyze and validate qualities of an architecture and drive architectural decision making. Following [12], we provide a set of activities (Fig. 3) to apply the EC perspective to the views.
1. Capture energy requirements; Requirements form the basis for change in relation to SA [1] and should be considered when strategical, economical or customer motives are present. For the case study we focused on DG's core functionality and investigate an activity encompassing the generation of 5000 documents, where each single document generation is considered a separate task. In relation to EC we formulate the requirement for DG to consume less energy while performing the specified task.
2. Create energy profile; An energy profile of the software provides the stakeholder with a starting point and a benchmark to evaluate results. The profile for DG was created with the following protocol; (1) Clear internal WUP memory, (2) Close unnecessary applications and services on the test server, (3) Start WUP and Perfmon measurements, (4) Perform specified task using client and (5) Collect and check Perfmon and WUP data from logging server. In total 22 measurements were performed divided over six series. After checking, 19 out of these 22 measurements were considered valid. For the energy profile, DG on average required 41 minutes and 49 seconds to generate the documents and with a *SEC* of 17560 Joule (J) (standard deviation 3577 J). An average *TEC* was found of 3.51 J ($\frac{17560}{5000}$) per generated document.

The AD for DG (Fig. 5), including the functional, concurrency and deployment view, learned that DG consists of the Document.exe, Config.exe and Connector.exe processes. The 'Generator' element (Document.exe) is responsible for the actual document generation, 'Utilities' (Config.exe) provides configuration options and the 'Composer', 'Interface' and 'Connector' elements (Connector.exe) handle communications.

[3] HP Proliant DL380 G5, Intel Xeon E5335 CPU, 800GB local storage (10.000 rpm), 64GB PC2-5300, 64 bit MS Windows Server 2008R2 (Restricted to 2 cores), VMware vSphere 5.1

Mapping the measurements on the AD, performance data shows a 49% *CPUU* of Document.exe, with an average utilization rate of 50.7% and 7.4% for the two available cores, whereas the other processes (Configuration.exe and Connector.exe) did not appear active. Consequently only the the *TEC* for Document.exe was added in the AD.
3. Assess against requirements; Using the energy profile an assessment should be performed on whether the software meets the requirements. Since we did not formulate a quantitative goal for the requirement, e.g. consume at most *X* Joule per document, we could not assess the requirements against the energy profile. Hence the profile was labeled as benchmark and we proceeded to determining adjustments.
4. Determine adjustments; Based on the assessment, the adjustments should be determined that are to be applied to the software or its context. From the previous activities we learned that the energy consumption during the activity was mainly caused by the 'Generator' element. Looking at the performance data, we argued that applying the *increase hardware utilization* tactic had the potential to let us meet the requirement. In collaboration with the DG developer the 'balancer' was added, operating according to the broker pattern, changing the SA as shown on the right hand side of Fig. 5.
5. Evaluate adjustments; After adjustment, an evaluation should be performed to determine whether the requirements are met and assure that no unwanted effects are brought about. After adjustment, 33 (out of 36) valid measurements were obtained (divided over seven series) following the earlier described protocol. On average the new version of DG required 39 minutes and 14 seconds to generate the documents with a *SEC* of 5782 J (std. dev. 1647 J). In this new situation, *TEC* was reduced with 67.1% to an average 1.16 J per generated document and a significant decrease in CPU activity was perceived (Fig. 4). The *CPUU* for Document.exe decreased to an average 19.2%, whereas the utilization rates of the cores appeared evenly divided (12.6% and 15.1% respectively). A note should be made though, as the database server was considered out of scope we did not include any effects on this hardware.

Threats to validity With regard to the validity of the case study, an evaluation is performed following the threats as identified in [13]. The *construct validity* considers whether the correct measures were identified for the object under study. With investigating EC, there is little discussion on the relevant measures. To relate these measurements to the software or elements thereof, established performance indicators were used; a common method that is also applied by others in this field of research.

In light of the *internal validity*, despite careful preparations, due to the behavior of services we can not be 100% certain that DG was solely responsible for the load on the test server. Therefore each individual measurement was checked for such processes

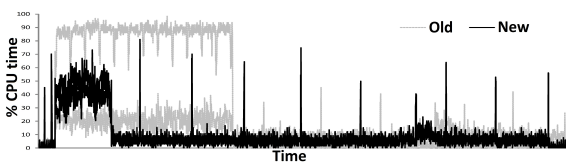

Fig. 4. Comparison of CPU activity of the test server during a measurement.

using performance data. Another threat is the lack of experience with configuring DG, e.g. we experienced firewall issues, resulting in a lower number of measurements at the start. For the evaluation we were more familiar with the case and relatively more valid measurements were obtained per series.

A threat to the *external validity* is the fact that the case study was performed in a separate test environment containing specific hardware. Given the relation between hardware and EC, different hardware could provide different findings in absolute terms. However, since an actual commercial software product was used, we argue that the proposed improvement is not specific to our environment.

Finally, *reliability* is concerned with the data and analysis thereof being dependent on the specific researchers. The measurements within the case study were performed by following a strict protocol, of which the activities are openly described. We therefore argue that following the described protocol should yield similar results.

4 Conclusion

In this paper we set out to investigate how EC can be positioned within the scope of software architecture through an EC perspective. In its current form the perspective enables stakeholders to identify, measure and analyze the EC of architectural elements, direct green efforts with regard their software product to where they are needed and verify the results. Using the perspective and the measures presented with the sustainability QA, a stakeholder has a means to quantitatively consider EC during the the design phase.

Alongside constructing the perspective, a case study was performed using a commercial software product (DG). The energy profile for DG directed our efforts and through an architectural change we reduced the energy consumption with 67.1% per generated document. Considering the frequency at which this task is performed, the savings could add up significantly from an organizational dimension.

However, we do acknowledge that the EC perspective is by no means as mature as other perspectives related to QAs. To further complete the perspective, among others by providing guidelines, more case studies are required for which the current perspective

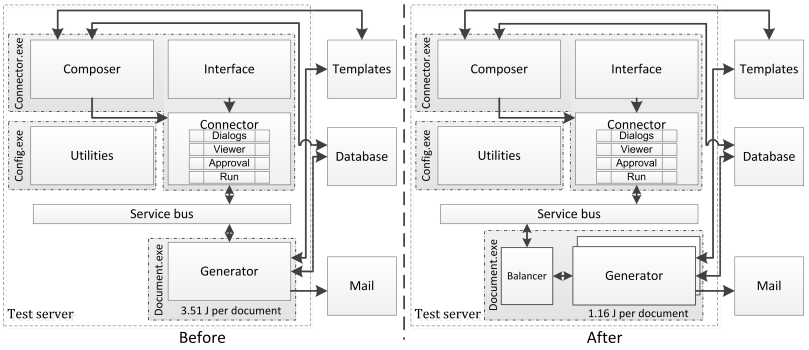

Fig. 5. Functional, concurrency and deployment view of DG for subsequent releases.

can serve as a starting point. Therefore the perspective should be considered as a step in the right direction to structurally consider the EC of a software product on SA level.

Based on the results presented in this paper, several directions for future research can be identified. First is a deeper investigation into the EC perspective and improvement by application in practice. For example improve on the visualization of EC aspects in the AD. Second is to investigate architecture variations, design patterns and tactics to find what actually comprises a sustainable software architecture. A final direction is to investigate, in depth, how insights gained from the architectural perspective can be translated to guidelines for software development.

References

1. L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. SEI Series in Software Engineering. Pearson Education, 2012.
2. P. Bozzelli, Q. Gu, and P. Lago. A systematic literature review on green software metrics. Technical report, Technical Report: VU University Amsterdam, 2013.
3. K. Grosskop and J. Visser. Identification of application-level energy optimizations. *Proceeding of ICT for Sustainability (ICT4S)*, pages 101–107, 2013.
4. G. Kalaitzoglou, M. Bruntink, and J. Visser. A practical model for evaluating the energy efficiency of software applications. In *ICT for Sust. 2014 (ICT4S-14)*. Atlantis Press, 2014.
5. E. Kern, M. Dick, S. Naumann, A. Guldner, and T. Johann. Green software and green software engineering–definitions, measurements, and quality aspects. *on Information and Communication Technologies*, page 87, 2013.
6. A. Kipp, T. Jiang, M. Fugini, and I. Salomie. Layered green performance indicators. *Future Generation Computer Systems*, 28(2):478 – 489, 2012.
7. P. Lago, R. Kazman, N. Meyer, M. Morisio, H. A. Müller, F. Paulisch, G. Scanniello, B. Penzenstadler, and O. Zimmermann. Exploring initial challenges for green software engineering: summary of the first greens workshop, at icse 2012. *ACM SIGSOFT Software Engineering Notes*, 38(1):31–33, 2013.
8. M. P. Mills. The cloud begins with coal: an overview of the electricity used by the global digital ecosystem. Technical report, Digital Power Group, August 2013.
9. S. Murugesan. Harnessing green it: Principles and practices. *IT Prof.*, 10(1):24–33, 2008.
10. A. Noureddine, R. Rouvoy, and L. Seinturier. Monitoring energy hotspots in software. *Automated Software Engineering*, pages 1–42, 2015.
11. G. Procaccianti, P. Lago, and G. A. Lewis. A catalogue of green architectural tactics for the cloud. In *Maint. and Evol. of Service-Oriented and Cloud-Based Systems (MESOCA), 2014 IEEE 8th Int'l Symp. on the*, pages 29–36, Sept 2014.
12. N. Rozanski and E. Woods. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives*. Addison-Wesley, 2011.
13. P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164, 2009.
14. Y. Sun, Y. Zhao, Y. Song, Y. Yang, H. Fang, H. Zang, Y. Li, and Y. Gao. Green challenges to system software in data centers. *Frontiers of Comp. Sc. in China*, 5(3):353–368, 2011.
15. S. te Brinke, S. Malakuti, C. Bockisch, L. Bergmans, and M. Akşit. A design method for modular energy-aware software. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1180–1182. ACM, 2013.
16. B. Zhong, M. Feng, and C.-H. Lung. A green computing based architecture comparison and analysis. In *Proc. of the 2010 IEEE/ACM Int'l Conf. on Green Computing and Communications & Int'l Conf. on Cyber, Physical and Social Computing*, pages 386–391. IEEE Computer Society, 2010.