Architectural Tactics for Energy Efficiency: Review of the Literature and Research Roadmap

Carlos Paradis
Information and Computer Sciences
University of Hawaii
cvas@hawaii.edu

Rick Kazman
Shidler College of Business
University of Hawaii
kazman@hawaii.edu

Damian A. Tamburri
Jheronimus Academy of Data Science
Technical University of Eindhoven
d.a.tamburri@tue.nl

Abstract

The energy consequences of software are rapidly growing: at the high-end, server farms consume enormous amounts of energy; at the low-end there is ever-increasing reliance on battery-powered mobile and Internet-of-Things (IoT) devices. However, there has been little attention to how software architectures can be designed for energy efficiency. While other software qualities such as performance or availability have been extensively studied, there is little research on how to reason about energy-consumption as a first-class quality. We provide a basis for reasoning about design decisions for energy efficiency by deriving a set of reusable architectural tactics derived from the research literature, via a taxonomic literature review. We used an open-search and snowballing methodology to obtain primary studies, and then used thematic coding to identify commonalities among the design strategies described. The result of this process is a taxonomy of 10 architectural tactics for energy efficiency. These tactics provide a rational basis for architectural design and analysis for energy efficiency.

1. Introduction

The environmental consequences of software are rapidly growing; for example, it is now a well-publicized fact that data centers account for the same amount of greenhouse gases as global aviation, and the energy requirements of data centers will only grow over time as our world becomes increasingly connected [1]. Furthermore, with our fixation on our smartphones, power consumption and battery-life are among the quality attributes that software engineers developing mobile applications must now worry about [2]. However, decades of research into software architecture cautions us that, to properly manage system quality attributes, an architectural approach is required. Simply put, coding alone is insufficient to address complex system-wide properties. Thus we claim that an architectural approach to energy efficiency is needed for software-intensive systems [3].

But the literature on the energy efficiency of software is fragmented. For example, Cloud systems typically do not have to be concerned with running out of energy (except in disaster scenarios), whereas this is a daily concern for users of mobile devices and some IoT devices. In cloud environments, on the other hand, scaling up and scaling down are core competencies and thus decisions must be made on a regular basis about optimal resource allocation. Finally, the ergonomics of mobile and IoT devices——their size, form factors, and heat output——constrain their design spaces. And the sheer number of IoT devices projected to be deployed in the future makes their energy usage a concern.

To address this research gap, in this article we derive a collection of architectural tactics—fundamental architectural design techniques—for energy efficiency. We do this through a taxonomic systematic literature review process, which employs a thematic coding technique borrowed from grounded theory to derive the taxonomy.

2. Architectural Tactics Explained

Tactics are fundamental design techniques that an architect can use to reason about and manage a quality attribute [4]. Tactics, like design patterns, are techniques that architects have been using for years. We do not *invent* tactics; we simply capture, classify, and catalog what architects actually have done in practice to manage quality attribute response goals.

Tactics are design decisions that influence the control of a quality attribute response. For example, if you want to design a system to have low latency or high throughput, there are a set of design decisions that you, as an architect, could make to achieve this. You could limit the rate of events (requests for service), or increase concurrency, or introduce priorities and scheduling, or limit the amount of processing that any single request receives, and so forth. A combination of these tactics

URI: https://hdl.handle.net/10125/71488
978-0-9981331-4-0

can help you to achieve responses that are produced within some time constraints.

Tactics are simpler than, and more primitive than design patterns. Tactics focus on the control of a single quality attribute response (although they may, of course, trade off this response with other quality attribute goals). Patterns, in contrast, typically focus on resolving and balancing multiple forces——multiple quality attribute goals. Thus we like to say, by way of analogy, that a tactic is an "atom" of design, whereas a pattern is a molecule. Consider Figure 1, a set of architectural tactics for performance [4]. The objectives in designing for performance are to "Control Resource Demand" and to "Manage Resources". An architect wanting to engineer a system with "good" performance needs to employ one or more of these options. That is the architect needs to decide if controlling resource demand is feasible, and if managing resources is feasible. In some systems the events arriving at the system can be managed, prioritized, or limited in some way, perhaps by filtering or aggregation. If this is not possible then the architect can only manage available resources in an attempt to generate responses within acceptable time constraints. For example, if an architect is designing a stock-trading system, dropping trades is never acceptable, and so the only option is to manage resources such that the system can keep up with peak performance demands. Within the "Manage Resources" category, an architect might choose tactics to Increase Resources, Introduce Concurrency, Maintain Multiple Copies of Computations, Maintain Multiple Copies of Data, and so forth. These tactics then, of course, need to be instantiated. As an example, an architect might choose the Half-sync/Half-async pattern as a way of introducing (and managing) concurrency [5] or might choose a load-balanced cluster deployment pattern to maintain multiple copies of computations [6].

To gain an understanding of the space of architectural design possibilities for energy efficiency, specifically the architectural tactics for energy efficiency, we pursued a taxonomic literature review strategy, as will be described next. Other approaches to gathering and validating tactics are, of course possible and have been pursued in the past such as expert interviews [7] and mining of patterns repositories [8].

Tactics provide a top-down way of thinking about design strategies, and this is useful in both design and analysis. A tactics categorization begins with a set of high-level design objectives related to the achievement of a quality attribute, and presents the architect with a set of (correspondingly high-level, abstract) options to choose from. These options then need to be further instantiated typically through some combination of patterns, frameworks, and code.

In Bass et al. [4] one can find tactics categorizations for the quality attributes of: availability, interoperability, modifiability, performance, security, testability, and usability. In addition we have created tactics for other quality attributes, such as DevOps [9].

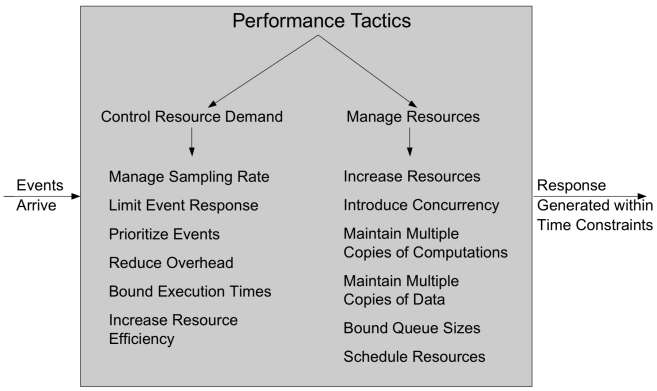

Figure 1. Performance Tactics

3. Research Design

To assess the state of the art in software-based approaches to energy efficiency, we first conducted a taxonomic systematic literature review using an "open search + snowballing" methodology [10].

For the open search, we used Engineering Village (Elsevier) and the databases Compendex and Inspec. The search query we used is as follows:

```
(
((”energy conservation” or ”energy utilization” or
”energy consumption” ”energy management” or
”energy efficiency” or ”energy efficient” or ”energy
saving”)
WN KY)
AND
((”software production” or ”software maintenance”
or ”software development” or ”software creation”)
WN KY)
)
AND
(English WN LA)
```

Our initial literature search resulted in 160 papers, and 128 after deduplication (99 from Compendex, 29 from Inspec), in addition to those identified via snowballing as potentially of interest, based on their titles, keywords, and abstracts. From these papers and the citations found therein we selected 71 papers for

detailed scrutiny and evaluation. Many of these selected papers were finally rejected as being irrelevant, trivial, off-topic, or redundant based on independent reading and consensus on the part of two of the authors. This resulted in a final dataset of 39 primary studies that received full scrutiny.

Subsequently, the primary study set was analysed using a tailored version of grounded-theory, enacting a process of open coding, where concepts found in the papers are given labels if and only if they would correspond to a specific tactic's definition (e.g., sections discussing the minimization of energy consumption in software code by reducing the occurrence of inter-module communication would be given the "Communication Siloing" tactic code). This was undertaken to determine candidate tactics, and to represent them with codes. Eventually we summarized those codes into categories. Each paper was read by two coders and any differences in the codes assigned were discussed and resolved. As per grounded-theory, the aforementioned process continued paper by paper, until theoretical saturation was reached, that is, when no more novel codes were discovered. The next section offers an overview of the results. We included, for each paper, a determination as to whether they were focused industrial problems. We found that all but one (paper 21 in Table 2) were purely research based, which is in agreement with paper 21 on the lack of industrially-relevant research in the literature.

3.1. Inter-Rater Reliability Assessment

To further substantiate the selection of papers and coding thereof, two observers were involved in triangulating the paper selection and coding using the well-known Krippendorff$_{Alpha}$ approach [11] to Inter-Rater Reliability assessment (IRR). The α score essentially measures a confidence interval score stemming from the agreement of values across two distinctly-reported observations about the same event or phenomenon. In our case the value was applied to measure the agreement between primary-study selection vectors as well as coding application vectors. The value was calculated initially to be 0.79 for the primary study selection, hence $\alpha < .800$, with 0.8 being a standard reference value for highly-confident triangulated observations. Subsequently, a discussion on the individual contrasting elements in the value calculations was used to drive the agreement between the two analyses up towards a sufficient alignment through discussion.Similarly, the IRR Score for the tactics coding eventually ended up to be much less initially ($\tilde{5}4\%$ agreement) connected to the varied nature of all potential tactics to be found in the target primary studies. Again, to address the misalignment, an instrumented point-by-point discussion was used to create a coherent merged list of tactics, as reported in the next section.

3.2. Energy Efficiency Tactics: Primary Study Descriptive Statistics

Altogether, the primary studies dataset yields a pretty positive picture for the field. On the one hand, Fig. 2 outlines the citation trends over time with every single dot indicating a paper instance and the Y-axis indicating how many citations does that particular paper have. The figure highlights indeed a slightly positive trend (trend-line in light blue on the figure) with the sector picking up pace as of the year 2012 and steadily gaining interest, with occasional outlier seminal papers, e.g., the work by Jing et al. [12].

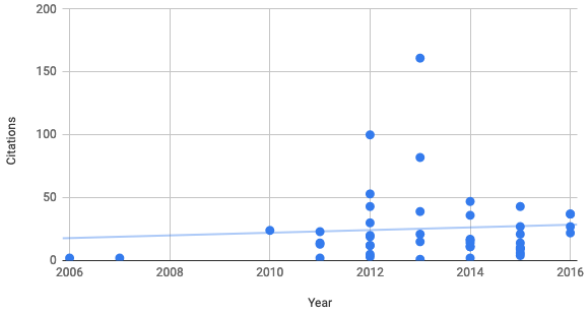

Figure 2. Primary Studies, chronometric citation trend.

At the same time, the type of venue in which results are being produced, showcased, and published (arranged between "conference" or "journal") indicates a relatively immature field, focusing primarily on disseminating results for the topic over conferences as opposed to more mature and consolidated journal contributions (see Fig. 3).

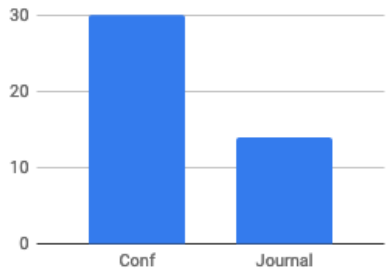

Figure 3. Primary Studies, venues trend.

4. Tactics for Energy Efficiency

The tactics categorization that emerged from our coding process is shown in Figure 4. Energy efficiency is, at its heart, about optimally utilizing resources. We therefore grouped the tactics into three broad categories: Resource Monitoring, Resource Allocation, and Resource Adaptation. As with existing tactics categorizations for performance or availability or modifiability these categories serve as a high-level checklist for a software architect or a reviewer. But the true design thinking goes into how those categories are refined into specific tactics and how those tactics are in turn translated into code, patterns, and components.

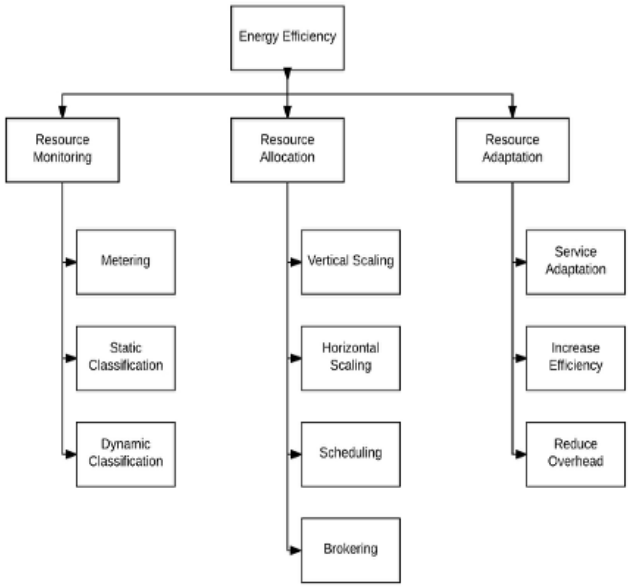

Figure 4. Discovered Tactics for Energy Efficiency.

We now examine each of those categories and their constituent tactics, in detail.

4.1. Resource Monitoring

The tactics for resource monitoring are Metering, Static Classification, and Dynamic Classification.

Metering: The Metering tactic involves collecting data about the energy consumption of computational devices, via a sensor infrastructure, in real time. At the coarsest level the energy consumption of an entire data center can be measured from its power meter. Individual servers or hard drives can be measured using external tools such as amp meters or watt-hour meters, or using built-in tools such as those provided with metered rack PDUs, ASICs (application-specific integrated circuits), etc.

Static Classification: There are cases where real-time data collection is infeasible. For example, if an organization is using an off-premise cloud, they might not have direct access to real-time energy data. In such cases we need to statically classify devices and computational resources, so that we can reason and make decisions about them in terms of resource allocation and adaptation. Such a classification might be based upon benchmarking or on reported device characteristics (such as a manufacturer's specifications).

Dynamic Classification: In cases where a static model of a device or computational resource is inadequate, a dynamic model might be required. Similar to the static classification tactic, dynamic classification is needed in cases where real-time data collection is infeasible. But, unlike static classification, dynamic models take into consideration transient conditions (such as workload) to determine energy consumption of a device or computational resource. The model could be a simple table lookup, a regression model based on data collected during prior executions, or a simulation.

4.2. Resource Allocation

The tactics for Resource Allocation are Vertical Scaling, Horizontal Scaling, Scheduling, and Brokering.

Vertical Scaling: Vertical scaling, also known as "scaling up" involves adding or activating resources to meet processing demands. For example, replacing a CPU with a faster version of the same CPU, or adding more memory to an existing server are examples of Vertical Scaling. However, in the context of energy efficiency, vertical scaling may be used to "scale down", that is removing or deactivating resources when demands no longer require them. Scaling down may involve spinning down hard drives, turning off CPUs, running CPUs at a slower clock rate, or shutting down current to blocks of the processor that are not in use.

Horizontal Scaling: Horizontal Scaling is a traditional tactic used to improve the performance of large-scale systems, such as server farms. The scaling is achieved by adding additional servers, VMs, or other resources to an existing pool of resources. For energy efficiency, horizontal scaling means the removal or idling of such resources. This may take the form of moving VMs onto the minimum number of physical servers (consolidation), combined with shutting down idle computational resources. In mobile applications horizontal scaling may be realized by sending part of the computation to the cloud.

Scheduling: Scheduling is the allocation of tasks to computational resources. In traditional operating systems the goal of scheduling may be to keep resources

as busy as possible, to allocate resources fairly among user tasks (which may have different priorities), or to achieve some quality of service, such as meeting deadlines. The point of scheduling in the context of energy efficiency is to optimize energy usage, given task constraints and respecting task priorities. Scheduling provides the scaling decisions for either horizontal or vertical scaling, based on data collected (using one or more Resource Monitoring tactics) about the state of the system being scheduled.

Brokering: A broker matches service requests (from clients) with service providers, supporting the identification and remote invocation of those services. Traditionally brokers make these matches based on a description of the service request (typically an API). However in the context of energy efficiency this request could be annotated with energy information, allowing the requestor to choose a service provider based on its (possibly dynamic) energy characteristics. For the cloud, this energy information could be stored in a "green service directory" populated by information from metering, static classification, or dynamic classification (the Resource Monitoring tactics). For a mobile device the information could be from an app store. Currently such information is ad-hoc at best, and typically non-existent in service APIs.

4.3. Resource Adaptation

The tactics for Resource Adaptation are Service Adaptation, Increase Efficiency, and Reduce Overhead.

Service Adaptation: Using the services of an energy broker (implementing the Brokering tactic) in a cloud context, or a controller in a multi-core context, a computational task can dynamically switch computational resources, such as service providers, to ones that offer better energy efficiency, or lower energy costs. For mobile devices this adaptation could be done by a human consumer, but in a cloud environment or in multi-core adaptation this function would typically be automated.

Increase Efficiency: Perhaps the most obvious tactic for resource adaptation is to Increase Efficiency. This is taught to every Computer Science major who takes a course in Data Structures and Algorithms. In this context, increasing efficiency is applied to improve the time or memory performance of critical algorithms and, in doing so, this increase of efficiency will also improve the energy efficiency of that software. However increasing efficiency may also be achieved by matching a service request to hardware that is best suited to fulfill that request. For example, if an algorithm can be parallelized this parallelization will only result in increased efficiency if it can be allocated to an environment that can host the required degree of parallelization.

Reduce Overhead: The use of intermediaries and abstractions (so important for modifiability) increases the resources consumed in processing an event stream. Hence removing these intermediaries typically improves latency and throughput. This is a classic modifiability/performance tradeoff. Separation of concerns, another foundation of modifiability, can also increase the overhead necessary to service an event if it leads to an event being serviced by an ensemble of components rather than a single component. The context switching and inter-component communication costs result in increased energy consumption, particularly when the components are on different nodes on a network. A strategy for reducing computational overhead and energy demands is therefore to co-locate resources and remove intermediaries and abstractions. Co-location may mean hosting cooperating components on the same processor to avoid the time delay of network communication or it may even mean putting the resources in the same execution container, to avoid even the expense of a method call and its context-switching costs.

5. Applying Tactics in Practice

A tactics categorization is a catalog of primitive design concepts and a framework for architectural reasoning. Additional work needs to be done to determine how best to use these tactics in real-world contexts and facing practical constraints. We now describe four broad application areas of how tactics have been used in practice in the past, and discuss how the energy efficiency tactics may be used in practice.

5.1. Using Tactics in Design and Code Reviews

How can the tactics for energy efficiency be best used in design and in reviews? Design and review contexts are the most common and perhaps most obvious uses for a collection of tactics. For example, in Cervantes et al. [9], a practice of using tactics as design analysis questionnaires is promoted. In this case, each tactic is turned into a question. An interviewer uses these questions to ask the architect if they considered using the tactic, how it is implemented, and the rationale for the way in which it was implemented or, if it was not implemented, the rationale for its exclusion.

In this way, in a very short time (approximately one hour per quality attribute reviewed) an analyst can gain a broad overview of the architectural approaches taken, or not taken, to address any quality attribute. We

thus propose that the tactics for energy efficiency can similarly be used to this end.

5.2. Using Tactics to Inform Design or Implementation Patterns

In addition, a tactics categorization could form the basis for a set of design or implementation patterns. To be clear, design patterns are not normally *invented*. Instead they *emerge* from the design successes and failures of many architects and architectures, over many systems, and over time. Existing books and web-sites of design patterns merely catalog what architects have already determined to be best practices in design for the quality attributes in question. Tactics can, however, provide guidance to creators of such patterns, e.g., multiple tactics can be combined and used to annotate or improve one or more patterns, to be jointly used in pursuit of a specific quality goal (e.g. [13], [4]).

Consider the Broker pattern[1], as shown in Fig. 5. This pattern, if naively implemented, would likely not be useful. The server shown in the diagram is a single point of failure, and a potential bottleneck for performance. There is no provision for security, such as authenticating requestors, and there are no test interfaces.

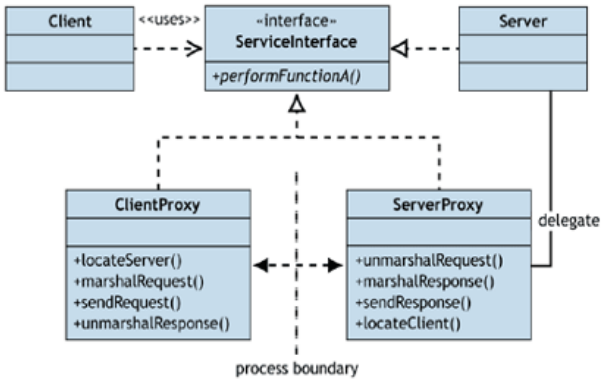

Figure 5. Broker Pattern from MSDN.

Thus the pattern is a skeleton, but in practice an architect could use tactics to improve this pattern so that it would be scalable, testable, robust in the face of failures, and so forth (as was done, for example, in [14]).

Sets of design patterns for energy efficiency have only recently begun to emerge. For example two recent papers describe sets of energy efficiency patterns for mobile devices [15] and embedded systems [16].

[1] https://msdn.microsoft.com/en-us/library/ff648096.aspx

5.3. Using Tactics to Create Reference Architectures

A reference architecture is a reference model mapped onto one or more architectural patterns. A mature reference architecture, like a design pattern, has been proven in business and technical contexts, and typically comes with a set of supporting artifacts that eases its use. For example, the Microsoft Application Architecture Guide [6] describes reference architectures for web applications, mobile applications, service applications, and so forth. An example reference architecture for web applications is shown in Figure **??**.

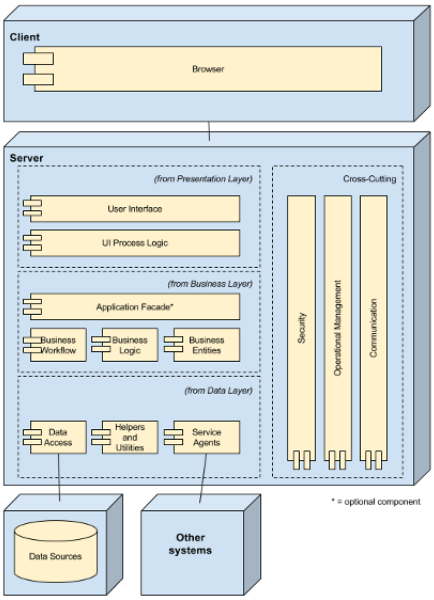

Figure 6. Web Application Reference Architecture

5.4. Creating "Best of Breed" Implementation Examples

Finally, a set of tactics need to eventually be realized in code. The majority of developers work best from examples, rather than abstractions. Thus it is imperative to have running code exemplifying as many tactics as possible, in as many different contexts as possible (operating system, cloud versus mobile, language of implementation, etc.).

Many example implementations of the tactics described here already exist. In the Appendix we have provided the full set of 39 references that were chosen for deriving the tactics, which include algorithms and implementation examples.

6. Discussion and Research Roadmap

A set of tactics, however well-motivated, is still quite abstract. This can be both a curse and a blessing. We dealt with one part of this challenge in the previous section: how to apply tactics in practice. But the abstraction has a benefit as well: it can stimulate creative thinking in terms of how tactics can be applied to improve the state of the art in software architecture. We now turn to some of these considerations.

6.1. Observations and Lessons Learned

There are a number of studies that could and should be undertaken to understand the use of these tactics and the tradeoffs between several crucial system quality attributes—energy efficiency, performance, availability, usability, and modifiability, and how these tradeoffs are affected by practical contextual factors.

For example, studies could implement tactics and measure resource usage, efficiency, and performance of code with and without employing a tactic. The impact of parallelism is a particular intriguing area for investigation, and there is little in the way of guidance for architects and developers at the moment in this technology area.

In terms of studying the tradeoffs, at the moment only the tradeoffs between performance and energy efficiency have been examined in any depth, as evidenced by our search of the literature. But energy efficiency additionally has implications for usability (if response times are slowed or screens are dimmed, or features are turned off), for availability (if backups are taken off-line and redundant resources are spun down, resulting in longer recovery times after failures), and for modifiability.

Amongst these quality tradeoffs, the greatest importance is to gain an understanding of the consequences of modularization (and buy versus build) decisions. This has received scant attention from the research community. For example, in [17] the authors reported on a study where they concluded that the use of frameworks and external libraries is detrimental to the energy efficiency of large applications.

In [9] the authors examined tradeoffs between latency, energy usage, and modifiability in mobile applications and showed that certain architectural choices in the implementation of the MVP pattern—bundling or dropping messages, realizing the Reduce Overhead tactic—can improve energy consumption by 30% without negatively impacting latency or modifiability. Similarly in [3] the authors demonstrated that, with some modest changes to communication protocols (realizations of the Reduce Overhead and Increase Efficiency tactics), an IoT application could reap 86% energy savings.

But these are just a few small studies, barely scratching the surface of the entire decision-space, and many important research questions remain. It is crucial to have a framework that enumerates the relevant contextual factors, and aids in reasoning about the consequences of modularization decisions and adoption of off-the-shelf components on energy efficiency. For example, we could study the effects of varying degrees of modularization, such as layering—and their effects on energy efficiency. But the tradeoff with modularity is just one dimension. We also want to know, for any level of modularity, the consequences on time-to-market and evolvability. In this way a project manager or architect could make reasoned decisions about such tradeoffs.

Thus, a framework that includes all relevant quality attributes—energy efficiency, performance, modularity, and so forth—is necessary to support an architect's reasoning. And such a framework can only be validated through empirical research.

6.2. Future Work: A Research Roadmap

There are a number of studies that should be undertaken to understand the architecture tradeoffs between several crucial system quality attributes—energy efficiency, performance, availability, usability, and modifiability—and how these tradeoffs are affected by practical contextual factors. Of these quality attributes, only the tradeoffs between performance and energy efficiency have been studied in any depth, as evidenced by our extensive search of the research literature. But energy efficiency additionally has implications for usability (if response times are slowed or screens are dimmed), for availability (if backups are taken off-line, resulting in longer recovery times after failures), and for modifiability.

Amongst these quality tradeoffs, the greatest importance is to gain an understanding of the consequences of modularization (and buy versus build) decisions. This has received scant attention from the research community. Capra et al. [17] reported on a study where they concluded that the use of frameworks and external libraries is detrimental to the energy efficiency of large applications. But, again, while these studies are a good start, many crucial research questions remain. It is important to have a framework that enumerates the relevant contextual factors, and aids in reasoning about the consequences of modularization decisions and adoption of off-the-shelf components on energy efficiency. So, for example,

we could study the effects of greater or lesser degrees of modularization–such as layering–and the correlation of modularity measures to energy efficiency. But the tradeoff with is just one dimension of the problem. We would also want to know, for any given level of modularity, the consequences on time-to-market and evolvability. In this way a project manager or architect could make reasoned decisions about such tradeoffs.

Thus a framework that includes all relevant quality attributes–energy efficiency, performance, modularity, and so forth–is necessary to support an architect's reasoning. And such a framework can only be validated through empirical research.

7. Conclusions

This paper has described a taxonomic systematic literature review process that we applied to the topic of energy efficiency in software-intensive systems, and the subsequent grounded-theory-based classification of the concepts found within the resulting papers. The point of this process was to derive a novel set of tactics for energy efficiency to address a research gap–that studies of energy efficiency of software are rather fragmented and no authoritative set of design approaches had thus far been described. Each of these tactics was defined and linked back to the papers from which they were derived.

Based on this set of tactics a number of real-world application contexts were described: using tactics in design and analysis (tactics-based questionnaires), creating and augmenting patterns, creating reference architectures, and creating "best of breed" implementation examples.

Finally, we provided a brief discussion on the need for experimental validation studies, and sketched the outline for a study on an area of research that is currently lacking, that is, understanding the tradeoffs between modularity and energy efficiency. It is hoped that this paper can serve as the foundation for a program of research, experimentation, and practice that will lead to a strong engineering foundation for the construction and evolution of systems that treat energy efficiency as a first-class quality attribute.

References

[1] D. M. Quan, A. Somov, and C. Dupont, "Energy usage and carbon emission optimization mechanism for federated data centers.," in *E2DC* (J. Huusko, H. de Meer, S. Klingert, and A. Somov, eds.), vol. 7396 of *Lecture Notes in Computer Science*, pp. 129–140, Springer, 2012.

[2] A. Frank, R. Asuncion, and M. Frank, "Smart optimization of energy consumption using iot.," in *MENACOMM*, pp. 1–6, IEEE, 2019.

[3] R. Kazman, S. Haziyev, A. Yakuba, and D. A. Tamburri, "Managing energy consumption as an architectural quality attribute.," *IEEE Software*, vol. 35, no. 5, pp. 102–107, 2018.

[4] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Addison-Wesley, 3rd ed., 2012.

[5] S. A. Chowdhury, A. Hindle, R. Kazman, T. Shuto, K. Matsui, and Y. Kamei, "Greenbundle: an empirical study on the energy impact of bundled processing.," in *Proceedings of ICSE '19*, pp. 1107–1118, IEEE / ACM, 2019.

[6] Microsoft, *Microsoft Application Architecture Guide*. USA: Microsoft Press, 2nd ed., 2009.

[7] J. Scott and R. Kazman, "Realizing and refining architectural tactics: Availability," 2009.

[8] J. Ryoo, P. A. Laplante, and R. Kazman, "Revising a security tactics hierarchy through decomposition, reclassification, and derivation.," in *SERE (Companion)*, pp. 85–91, IEEE, 2012.

[9] H. Cervantes and R. Kazman, *Designing Software Architectures: A Practical Approach*. Addison-Wesley Professional, May 2016.

[10] S. Jalali and C. Wohlin, "Systematic literature studies: database searches vs. backward snowballing.," in *ESEM* (P. Runeson, M. Höst, E. Mendes, A. A. Andrews, and R. Harrison, eds.), pp. 29–38, ACM, 2012.

[11] K. Krippendorff, *Content Analysis: An Introduction to Methodology*. Beverly Hills, CA: Sage Publications, Inc., 1980.

[12] S. Jing, S. Ali, K. She, and Y. Zhong, "State-of-the-art research study for green cloud computing," *The Journal of Supercomputing*, vol. 65, pp. 445–468, 2011.

[13] N. Harrison and P. Avgeriou, "How do architecture patterns and tactics interact? a model and annotation," *Journal of Systems and Software*, vol. 83, no. 10, pp. 1735–1758, 2010.

[14] N. Harrison and P. Avgeriou, "Incorporating fault tolerance tactics in software architecture patterns," in *SERENE '08: Proceedings of the 2008 RISE/EFTS Joint International Workshop on Software Engineering for Resilient Systems*, pp. 9–18, ACM, 2008.

[15] L. Cruz and R. Abreu, "Catalog of energy patterns for mobile applications," *Empirical Software Engineering*, vol. 24, pp. 2209–2235, 2019.

[16] M. Schaarschmidt, M. Uelschen, E. Pulvermuellerm, and C. Westerkamp, "Framework of software design patterns for energy-aware embedded systems," in *Proceedings of the 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2020)*, 2020.

[17] E. Capra, C. Francalanci, and S. Slaughter, "Measuring application software energy efficiency.," *IT Prof.*, vol. 14, no. 2, pp. 54–61, 2012.

8. Appendix

Table 1. References.

ID	Reference	Industry
1	Sarah Abdulsalam, Ziliang Zong, Qijun Gu, Meikang Qiu. "Using the Greenup, Powerup, and Speedup metrics to evaluate software energy efficiency". In Proceedings of the Sixth International Green and Sustainable Computing Conference (IGSC), 2015.	No
2	Nadine Amsel, Zaid Ibrahim, Amir Malik, Bill Tomlinson. "Toward sustainable software engineering". Proceedings of the 33rd International Conference on Software Engineering (ICSE), 2011, pp. 976-979.	No
3	Luca Ardito, Maurizio Morisio, "Green IT – Available data and guidelines for reducing energy consumption in IT systems". Sustainable Computing: Informatics and Systems, 4, 1, March 2014, pp. 24-32.	No
4	L. Ardito, G. Procaccianti, M. Torchiano A. Vetrò, "Understanding Green Software Development: A Conceptual Framework," IT Professional, 17, 1, Jan.-Feb. 2015, pp. 44-50.	No
5	A. Bhardwaj S. Saurav, "A portable platform to estimate power consumption of software modules," International Conference on VLSI Systems, Architectures, Technology and Applications (VLSI-SATA), 2016, pp. 1-6.	No
6	P. Bartalos M. B. Blake, "Green Web Services: Modeling and Estimating Power Consumption of Web Services". IEEE 19th International Conference on Web Services, 2012, pp. 178-185.	No
7	T. W. Bartenstein Y. David Liu, "Green Streams for data-intensive software," 35th International Conference on Software Engineering (ICSE), San Francisco, CA, 2013, pp. 532-541.	No
8	Suparna Bhattacharya, K. Gopinath, Karthick Rajamani, Manish Gupta. "Software Bloat and Wasted Joules: Is Modularity a Hurdle to Green Software?". Computer 44, 9, September 2011, pp. 97-101.	No
9	E. Capra, C. Francalanci S. A. Slaughter, "Measuring Application Software Energy Efficiency," in IT Professional, 14, 2, March-April 2012, pp. 54-61.	No
10	Eugenio Capra, Chiara Francalanci, Sandra Slaughter. "Is software 'green'? Application development environments and energy efficiency in open source applications". Information and Software Technology 54, 1, January 2012, pp. 60-71.	No
11	Kayun Chantarasathaporn Chonawat Srisa-an. "Energy conscious factory method design pattern for mobile devices with C# and intermediate language". Proceedings of the 3rd international conference on Mobile technology, applications & systems, Article 29, 2006.	No
12	Samuel Chinenyeze, et al. "An Aspect Oriented Model for Software Energy Efficiency in Decentralised Servers." International Conference on ICT for Sustainability (ICT4S), 2014.	No
13	P. G. Flikkema, K. R. Yamamoto S. Boegli, "Starting from green: Energy-centric transformation of smart object architectures," IEEE Globecom Workshops, 2012, pp. 396-400.	No
14	Carlo Fragni, Luís Henrique Maciel Kosmalski Costa, "ECO–ALOC: Energy-efficient resource allocation for cluster-based software routers", Computer Networks, 56, 9, June 2012, pp. 2249-2261.	No
15	A. Hindle, "Green Software Engineering: The Curse of Methodology," IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2016, pp. 46-55.	No
16	Abram Hindle, Alex Wilson, Kent Rasmussen, E. Jed Barlow, Joshua Charles Campbell, Stephen Romansky. "GreenMiner: a hardware based mining software repositories software energy consumption framework". Proceedings of the 11th Working Conference on Mining Software Repositories (MSR), 2014, pp. 12-21.	No
17	Si-Yuan Jing, Shahzad Ali, Kun She, Yi Zhong. 2013. "State-of-the-art research study for green cloud computing". Journal of Supercomputing 65, 1, July 2013, pp. 445-468.	No
18	M. A. Khan, C. Hankendi, A. K. Coskun, M. C. Herbordt. "Software optimization for performance, energy, and thermal distribution: Initial case studies", Proceedings of the International Green Computing Conference and Workshops (IGCC), 2011, pp. 1-6.	No
19	Sedef Akınlı Koçak, Gülfem Işıklar Alptekin, Ayşe Başar Bener, "Evaluation of Software Product Quality Attributes and Environmental Attributes using ANP Decision Framework", Third International Workshop on Requirements Engineering for Sustainable Systems, 2014.	No
20	Andreas Litke, Kostas Zotos, Alexander Chatzigeorgiou, George Stephanides, "Energy Consumption Analysis of Design Patterns", Proc. World Academy of Science, Engineering and Technology, 6, June 2005.	No

Table 2. References (continued).

ID	Reference	Industry
21	Irene Manotas, Christian Bird, Rui Zhang, David Shepherd, Ciera Jaspan, Caitlin Sadowski, Lori Pollock, James Clause. "An empirical study of practitioners' perspectives on green software engineering". Proceedings of the 38th International Conference on Software Engineering (ICSE), 2016, pp. 237-248.	Yes
22	A. Markiewicz, P. N. Tran A. Timm-Giel, "Energy consumption optimization for software defined networks considering dynamic traffic," 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet), 2014, pp. 155-160.	No
23	Adel Noureddine, Romain Rouvoy, Lionel Seinturier. 2015. "Monitoring energy hotspots in software". Automated Software Engineering, 22, 3 (September 2015), pp. 291-332.	No
24	Adel Noureddine, Aurelien Bourdon, Romain Rouvoy, Lionel Seinturier. "Runtime monitoring of software energy hotspots". Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2012, pp. 160-169.	No
25	Candy Pang, Abram Hindle, Bram Adams, and Ahmed E. Hassan. "What Do Programmers Know about Software Energy Consumption?" IEEE Software 33, 3, May 2016, pp. 83-89.	No
26	Paula Petrica, Paula, Adam Izraelevitz, David Albonesi, Christine Shoemaker, "Flicker: A dynamically adaptive architecture for power limited multicore systems", ACM SIGARCH Computer Architecture News, 41, 3, 2013.	No
27	Giuseppe Procaccianti, Stefano Bevini, Patricia Lago, "Energy Efficiency in Cloud Software Architectures", 27th Conference on Environmental Informatics, 2013, pp. 291-299.	No
28	Giuseppe Procaccianti, Héctor Fernández, Patricia Lago. 2016. "Empirical evaluation of two best practices for energy-efficient software development", Journal of Systems and Software 117, July 2016, pp. 185-198.	No
29	Giuseppe Procaccianti, Patricia Lago, Stefano Bevini, "A systematic literature review on energy efficiency in cloud software architectures", Sustainable Computing: Informatics and Systems, 7, September 2015, pp. 2-10.	No
30	G. Procaccianti, P. Lago G. A. Lewis, "A Catalogue of Green Architectural Tactics for the Cloud," 2014 IEEE 8th International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems, 2014, pp. 29-36.	No
31	C. Sahin, F. Cayci, J. Clause, F. Kiamilev, L. Pollock K. Winbladh, "Towards power reduction through improved software design," 2012 IEEE Energytech, 2012, pp. 1-6.	No
32	Cagri Sahin, Furkan Cayci, Irene Lizeth Manotas Gutiérrez, James Clause, Fouad Kiamilev, Lori Pollock, Kristina Winbladh, "Initial explorations on design pattern energy usage", Proceedings of the First International Workshop on Green and Sustainable Software (GREENS), 2012, pp. 55-61.	No
33	Clauirton Siebra, Paulo Costa, Regina Miranda, Fabio Q. B. Silva, Andre Santos. "The software perspective for energy-efficient mobile applications development". Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia (MoMM '12), 2012, pp. 143-150.	No
34	Nidhi Singh Shrisha Rao. "Meta-learning based architectural and algorithmic optimization for achieving green-ness in predictive workload analytics". Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC '13), 2013, pp. 1169-1176.	No
35	Jasmeet Singh, Kshirasagar Naik, Veluppillai Mahinthan, "Impact of Developer Choices on Energy Consumption of Software on Servers", 2015 International Conference on Soft Computing and Software Engineering, Procedia Computer Science 62, 2015, pp. 385 – 394.	No
36	Yuzhong Sun, Yiqiang Zhao, Ying Song, Yajun Yang, Haifeng Fang, Hongyong Zang, Yaqiong Li, Yunwei Gao. "Green challenges to system software in data centers". Frontiers if Computer Science China 5, 3 (September 2011), pp. 353-368.	No
37	Ingolf Waßmann, Daniel Versick, Djamshid Tavangarian. "Energy consumption estimation of virtual machines". Proceedings of the 28th Annual ACM Symposium on Applied Computing (SAC), 2013, pp. 1151-1156.	No
38	Chenlei Zhang Abram Hindle. "A green miner's dataset: mining the impact of software change on energy consumption." 11th Working Conference on Mining Software Repositories (MSR), 2014.	No
39	Benjamin Zhong, Ming Feng, Chung-Horng Lung. "A Green Computing Based Architecture Comparison and Analysis". Proceedings of the IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing (GREENCOM-CPSCOM), 2010, pp. 386-391.	No