Leveraging Model-Driven Engineering for Energy Optimization

Thibault Béziers La Fosse

MathSTIC

Spécialité : STIC
Laboratoire : LS2N
Equipe : NaoMod

Leveraging Model-Driven Engineering for Energy Optimization

Thibault Béziers la Fosse
Mél : thibault.beziers-la-fosse@univ-nantes.fr

Energy consumption is a critical point when developing applications. Either for battery-saving purposes, for lowering the cost of data-centers, or simply for the sake of having an eco-friendly program, reducing the energy needed to run a software becomes mandatory. Model-Driven Engineering has shown great results when it comes to program understanding and refactoring. Modeling the source code along with its energy consumption could be a powerful asset to programmers in order to develop greener code. For that purpose, this paper presents an approach for modeling energy consumption inside a source code model. Energy metrics are gathered at runtime, modeled using the standard Structured Metrics Meta-model, and associated to the source code model, enabling model-driven techniques for energy analysis and optimizations.

1 Introduction

Energy Consumption becomes a major concern when developing software. The massive electricity needs of data-centers represented 1.8% total U.S. consumption in 2016, and an estimated account for about 2% of global greenhouse gas [1–3]. Furthermore, the expense to power and cool the data-centers escalated to a significant cost factor: for every $1.00 spent on new hardware, an additional $0.50 is spent on power and cooling [4]. Those factors severely limits the expansion of IT resources. Reducing the consumption of software systems has thus become an important ecological, economical, and technological challenge.

Traditionally, such concerns tend to be addressed at the lower levels (hardware or middleware), and several strategies have been presented in the past years towards that purpose [5]. However, promising results have been introduced using software-level energy management approaches [6], which, combined with lower level optimizations, can effectively reduce the energy consumption of programs.

Unlike hardware-level energy saving techniques, software-level approaches often need the implication of developers. In fact, to develop a *greener* code, a certain level of energy-awareness is helpful to make relevant design choices. A study showed that more than 80% of programmers did not consider the energy consumption when developing softwares, even if they know how important it can be, especially in the realm of mobile application development [7, 8]. We believe that a representation of a source code, along with energy consumption would help developers making the right design choices.

In the realm of Model-Driven Engineering (MDE), this representation is a model, conforming to a meta-model describing its syntax and semantic [9]. Such model is typically used during the software development, as a specification model to generate a source code, or as a source code model, reverse-engineered from the software in order to analyze or refactor it.

This paper proposes a MDE approach for modeling the source code of a software and characterizing it with energy measurements. A first model of this software is generated with the static reverse-engineering framework MoDisco [10]. In addition we propose a framework dynamically gathering energy measurements, at runtime.

Those measurements are persisted in a second model conforming to the Structured Metrics Meta-model[1] (SMM). In order to characterize the source code with energy metrics, the two models are associated: elements in the source code (e.g., Java methods) are linked to the energy measurements, enabling the analysis of the source code energy consumption.

This paper is organized as follows: Section 2 presents our approach, Section 3 discusses the approach, and Section 4 concludes the paper.

2 Approach

We detail in this section the several steps necessary to characterize our source code model with energy measurements. First, the source code model is statically built using the MoDisco reverse-engineering

[1] https://www.omg.org/spec/SMM/

```java
package mainPackage;
public class App {
    public static void main(String[] args) {
        foo();
    }
    public static void foo() {
        //do something
    }
}
```

Figure 1: Simple Java program

framework. Second, the code is instrumented, in order to add probes inside the program, which is then executed to gather the energy measurements at runtime. Third, the measurements are persisted in a model conforming to the Structured Metrics Meta-model, and associated with the MoDisco source code model. The Java program in Figure 1 is used to describe our approach. Our application is available on GitHub[2].

2.1 Source Code Model

The MoDisco framework [10] is designed to enable program analysis, by creating a static model of a source code using reverse-engineering. Considering a source code written in Java, MoDisco generates the corresponding model conforming to a Java meta-model. This model only represents static aspects of the code, such as classes, methods, statements or expressions. The energy measurements have to be dynamically gathered, and are not available in the initial MoDisco model. Nonetheless, a benefit of MDE is that such measurements can be modeled and then associated with our source code model, as presented in the next sections. Applying MoDisco on the program in Figure 1 would produce a model containing all the source-code elements of such program.

2.2 Energy Measurements Computation

Once the source code model is generated, the energy is measured. In order to get such measurements for each *method* in the program, it has to be instrumented. We use the ASM instrumentation library[3] for that purpose. The JVM byte-code is visited, and probes are added in the byte-code of every method of the program. A first probe is added in every method entry point, traces down the method call, gets the system time, and finally gets the energy consumed by the CPU at the specific moment, in microjoule.

Another probe is added at every method exit point. This second probe computes the duration of this method execution, and the energy it has consumed. To do so, the energy consumed by the CPU is fetched a second time, and subtracted to the value obtained in the first probe. Our energy measurements are performed using the same approach as `jRAPL` [6].

Running the instrumented code triggers the probes to gather measurements. For each method, the following metrics are obtained:

1. Its timestamp
2. The energy it consumed
3. The methods that have been called inside.

Running the code can be done either by executing the main method of a program, or through test cases. During the execution, the measurements are published on a messaging queue, in order to be analyzed after the execution. Indeed, analyzing the traces is an expensive operation, and performing it after the execution limits the overhead induced by the execution of the instrumented statements.

Furthermore, our approach is relying on existing energy measurement tools, and do not aim at improving their accuracy. Performing the measures causes an important workload, limiting the accuracy of the energy measurements [11]. For that reason, we can only provide the energy consumed at the method level, or at coarser granularities.

[2] https://github.com/atlanmod/EnergyModel
[3] https://asm.ow2.io/

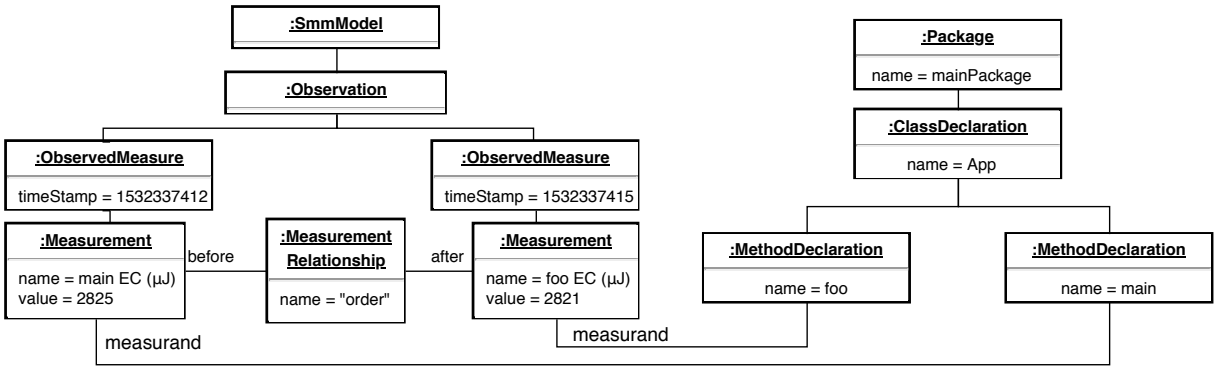

Figure 2: Excerpt of the source code model (the `MethodDeclaration` instances) characterized with energy measurements

2.3 Energy Measurements Modeling

In this main step of our approach, we persist the energy measurements into a model conforming to SMM and associate it to the source code model that MoDisco generates. This is performed by reading the measurements sent in the messaging queue one by one. Energy consumed, timestamps and internal method calls are persisted as elements in the model. Finally the measurements are linked to the methods from the source-code level, hence characterizing those.

In Figure 2, the energy consumed is contained in the `Measurement` elements, as values. The method calls are represented using the `MeasurementRelationship`. The "before" and "after" links point towards the `Measurement` elements, describing in which order the methods have been called. Here for instance, `main()` calls `foo()`.

Furthermore, the timestamps of the methods invocations are available in the `ObservedMeasure` elements. Finally the source code model elements (e.g. `MethodDeclaration`) are associated to the `Measurement` using the link labeled as `measurand`.

3 Discussion

This source code model characterized with energy measurements offers developers a better energy-awareness than standard software engineering techniques: First, an holistic model of the program is created. The association between MoDisco's Java meta-model and SMM offers a representation of the source code characterized with several dynamic information, available for analysis purposes. Second, the program can be refactored and optimized only by working at the model level.

Programmers can specify transformation rules for refactoring [12], without having to go back to the source code level, in order to optimize the energy consumption using existing energy-efficient practices [13–16]. As stated by G. Pinto et al., refactoring approaches focusing on improving the energy consumption of programs are lacking [17]. Our model could help programmers refactoring their applications towards that purpose, by specifying a model transformation to apply to the source-code model, and generating the program optimized [18]. The energy measurements present in the model can be used as predicates for the transformation. For instance, a model transformation could be specified, only targeting the methods that consume more than a specified energy. This transformation would change all the existing Doubles to Floats, as it has been shown that it can reduce the energy consumption [6]. The modified source code of the program can then be generated again, using MoDisco code generator, and be used to consume less energy.

4 Conclusion and Future Work

This paper presents an approach for modeling the source code of an application and decorating it with energy measurements. Those measurements are dynamically gathered, modeled using SMM, and associated with a source code model statically generated by MoDisco. This model can be used by programmers as a basis for analyzing and optimizing programs, and offers a better energy-awareness, useful for implementing greener applications.

References

[1] Gary Cook. How clean is your cloud. *Catalysing an energy revolution*, page 11, 2012.

[2] Molly Webb et al. Smart 2020: Enabling the low carbon economy in the information age. *The Climate Group. London*, 1(1):1–1, 2008.

[3] Arman Shehabi, Sarah Smith, Dale Sartor, Richard Brown, Magnus Herrlin, Jonathan Koomey, Eric Masanet, Nathaniel Horner, Inês Azevedo, and William Lintner. United states data center energy usage report. 2016.

[4] ME Jed Scaramella and Matthew Eastwood. Solutions for the datacenter's thermal challenges. *IDC, January*, 2007.

[5] Sparsh Mittal. A survey of techniques for improving energy efficiency in embedded computing systems. *International Journal of Computer Aided Engineering and Technology*, 6(4):440–459, 2014.

[6] Kenan Liu, Gustavo Pinto, and Yu David Liu. Data-oriented characterization of application-level energy optimization. In *International Conference FASE'15*. Springer.

[7] Irene Manotas, Lori Pollock, and James Clause. Seeds: a software engineer's energy-optimization decision support framework. In *Proceedings of the 36th International Conference on Software Engineering*, pages 503–514. ACM, 2014.

[8] Candy Pang, Abram Hindle, Bram Adams, and Ahmed E Hassan. What do programmers know about software energy consumption? *IEEE Software*, 33(3):83–89, 2016.

[9] Jean Bézivin. On the unification power of models. *Software & Systems Modeling*, 4(2):171–188, 2005.

[10] Hugo Bruneliere, Jordi Cabot, Frédéric Jouault, and Frédéric Madiot. Modisco: a generic and extensible framework for model driven reverse engineering. In *Proceedings of the IEEE/ACM International Conference ASE'10*. ACM.

[11] Suz anne Rivoire, Mehul A Shah, Parthasarathy Ranganathan, and Christos Kozyrakis. Joulesort: a balanced energy-efficiency benchmark. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 365–376. ACM, 2007.

[12] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[13] Rui Pereira, Marco Couto, João Saraiva, Jácome Cunha, and João Paulo Fernandes. The influence of the java collection framework on overall energy consumption. In *GREENS'16*. ACM, 2016.

[14] Giuseppe Procaccianti, Héctor Fernández, and Patricia Lago. Empirical evaluation of two best practices for energy-efficient software development. *Journal of Systems and Software*, 117:185–198, 2016.

[15] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. Enerj: Approximate data types for safe and general low-power computation. In *ACM SIGPLAN Notices*, volume 46, pages 164–174. ACM, 2011.

[16] Wellisson GP da Silva, Lisane Brisolara, Ulisses B Correa, and Luigi Carro. Evaluation of the impact of code refactoring on embedded software efficiency. In *Proceedings of the 1st Workshop de Sistemas Embarcados*, pages 145–150, 2010.

[17] Gustavo Pinto, Francisco Soares-Neto, and Fernando Castor. Refactoring for energy efficiency: a reflection on the state of the art. In *Proceedings of the Fourth International Workshop on Green and Sustainable Software*, pages 29–35. IEEE Press, 2015.

[18] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl: A model transformation tool. *Science of computer programming*, 72(1-2):31–39, 2008.