Energy-Efficient Code by Refactoring

Marion Gottschalk, Jan Jelschen, Andreas Winter
Carl von Ossietzky Universität, Oldenburg, Germany
{gottschalk, jelschen, winter}@se.uni-oldenburg.de

Abstract

The rising number of mobile devices increase the interest in longer battery durations. To increase battery duration, researchers try to improve e.g. different hardware components, such as processors and GPS for lower energy consumption. Frequently, software optimization possibilities to save energy are forgotten. Hence, an approach is shown to reduce energy consumption of applications by reengineering. Therefor, energy-wasteful code in applications is searched by code analysis and then restructured to optimize their energy consumption. Energy savings are validated by different energy measurements techniques.

1 Motivation

The scope of mobile devices is increasing, due to new user requirements, more powerful processors, and diversity of applications. Mobile devices are used for many tasks, and thus, users carry their smartphone with them the whole day. Hence, batteries of these devices need to endure this time. The Blackberry Z10 shows the contrary [1]: First tests showed that batteries of Z10s are empty after about five hours. These are only 21 % of a day which does not match users expectation's of at least one day uptime. Blackberry's solution is an external battery which loads the main battery to lengthen its operating time. This is not acceptable for most users to use two batteries. Therefore, manufacturers should be interested in energy-efficient software to extend devices' battery duration.

To improve batteries' duration by software optimization, software evolution techniques are used: *Reverse Engineering* produces an abstract view of components and relationships inside on applications and provides to analyze for energy wasting software behavior [2]. *Reengineering* is applied to improve existing applications regarding their energy consumption, in which the intended application behavior is not changed. Therefor, code analyzing and restructuring are used to obtain more energy-efficient code. Both steps describe *refactoring* which is concerned with detecting and restructuring inefficient code (code smells) without changing its behavior [3].

This paper is structured as follows: First, some energy code smells which have been validated to be energy-wasteful by measurement are presented in Section 2. Next, the process of identifying and restructuring of energy code smells is described in Section 3. Section 4 shows a validity of the energy consumption of applications before and after reengineering. Section 5 concludes this paper with an outlook.

2 Energy Refactoring

Energy refactorings contain energy code smells which are energy-wasteful parts of code. These energy code smells and their restructurings are defined and described similar as the code smells of Fowler [3]. Two such energy code smells are *Binding ressources too early* and *Third party advertising* [4].

Binding ressources too early: When applications are binding resources it could be that applications request a resource too early. This means that resources, like GPS and WiFi, are started and consume energy before they are really needed by the application. Detecting *Binding resources too early* requires to identify the position of methods in which hardware resources are started. At this point, programmers have to check the program behavior to decide whether hardware resources are really required or not. Unnecessary method calls must be moved to a more appropriate position, so that hardware resources start only when they are really needed. It has be considered that other code parts may also have to be shifted together with the code, e.g. dependents statements.

Third party advertising: Another aspect are advertisings in free Android applications. Pathak et al. [5] show that third party advertising is often the biggest energy consumer of an application, even so it is not necessary to execute the application. The energy consumption may be, e.g. due to WIFI connections, which updates the advertising every few minutes. This energy code smell can be detected when it is known which API for adverts is used. If source code for advertising is detected and no legal constraints exist, it can simply be removed.

More energy code smells and its restructurings are defined in Gottschalk et al. [4].

3 Reengineering Process

Reengineering towards energy-efficient code is a process which forms the basics for a semi-automatize reengineering process to save energy in applications. The four steps *parse*, *reverse engineering*, *restructuring*, and *unparse* are shwon in Figure 1.

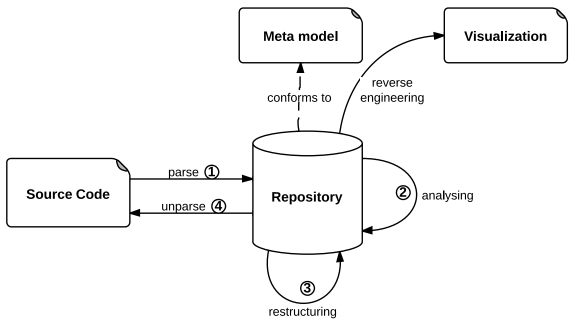

Figure 1: Model based Reengineering Process

Firstly, code must be parsed into a higher unified abstraction (reverse engineering) to enable an efficient code analysis. Therefor, a graph structured repository which conforms to a meta model are used (cf. e.g. Ebert et al. [6]). The meta model defines a clear, precise, and targeted structure and documentation of underlying data structures which enable code analysis on different applications [7]. Secondly, the abstracted code is analyzed, e.g by static code analysis which has proved in software evolution, to identify energy code smells when methods, such as methods to display advertisings, are known. Thirdly, the energy-wasteful parts are changed by a restructuring on the repository to remove the energy code smells. Lastly, the graph is unparsed to code so that it can be executed as application on mobile devices. During this process it is also possible to visualize the abstracted code.

The first and fourth steps are realized by the SOAMIG parser [8], and hence, these steps do not need further attention in this case. Due to the usage of the SOAMIG parser, TGrpahs which conform to a Java meta model are generated and saved into the repository. TGraphs are directed graphs, whose nodes and edges are typed, attributed, and ordered [9]. Techniques of the TGraph approach are used for the second and third step to execute queries and instructions on TGraphs. Both steps represent the above described refactoring which must be implemented. The TGraph approach contains amongst others GReQL (Graph Repository Query Language) [9] and GReTL (Graph Repository Transformation Language) [9] which are applied in this process. The implementation of both is realized by the JGraLab API [10]. GReQL is used for the second step to identify energy code smells on TGraphs. In third step GReTL is used to transform the TGraph so that more energy-efficient code can be generated in the last step.

4 Validation

To check the benefits of energy refactorings the energy measurement by Schröder [11] is used. First measurements are made with the application GPSPrint [12] regarding to the energy code smell *Binding resources too early*. This reengineering was done partly manually so that first results of energy saving are visible. One energy measurement is shown in Figure 2.

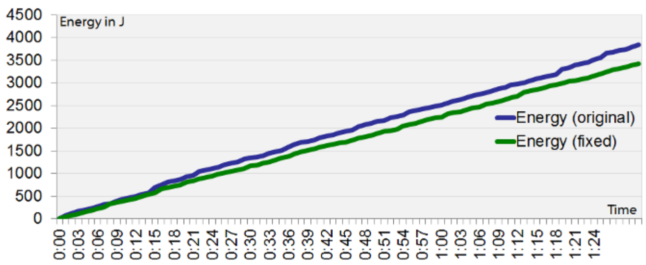

Figure 2: Energy measurement

This measure shows that approximately 405 J in 1:30 h were saved after reengineering (fixed graph). The energy measurement based on Android API methods which deliver different values about the battery status. To calculate the energy consumption three measuring techniques were realized: Delta-B, analysis of system files, and energy profiles [11]. *Delta-B* compares changes of battery charge level with battery capacity to measure energy. *Energy profiles* are provided by devices manufacturer who saves energy information of hardware components in an internal XML-file. In Figure 2 *analysis of system files* is used. This measure saves information about battery status in a readable system file of Android devices. Due to that, the actually battery voltage and discharge rate per minute are known, and the energy consumption can be calculated. This energy measure which reads an system file are not as exactly as a directly measurement on hardware, so that this results give only a trend of energy savings [11]. Also, it must be considered that reading a XML-file costs energy so that the number of accesses should be small as possible.

5 Outlook

First results show that it is possible to save energy by software optimization. In next steps, the energy refactoring *Third party advertising* is going to be implemented, and then, new energy measurements are going to be executed. Thereby, more measurements of same use cases are going to be made to identify the average energy savings of applications. At the end, a refactoring catalog and more energy measure techniques are developed.

References

[1] M. Kremp. (2013, January) Blackberry Z10 im Test: Handy mit Nottank. Spiegel online. http://www.spiegel.de/netzwelt/gadgets/angefasst-der-blackberry-z10-im-test-a-880411.html

[2] E. J. Chikofsky, J. H. Cross, "Reverse Engineering and Design Recovery: A Taxonomy," *Software, IEEE*, 1990.

[3] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, *Refactoring: Improving the Design of Existing Code*, Addison Wesley, 2002.

[4] M. Gottschalk, M. Josefiok, J. Jelschen, A. Winter, "Removing Energy Code Smells with Reengineering Services," in *Lecture Notes in Informatics*. GI, 2012.

[5] A. Pathak, Y. Charlie Hu, M. Zhang, "Fine Grained Energy Accounting on Smartphones with Eprof," in *EuroSys'12*, 2012.

[6] J. Ebert, V. Riediger, A. Winter, "Graph Technology in Reverse Engineering, The TGraph Approach," in *10th Workshop Software Reengineering*, GI, 2008, pp. 67–81.

[7] D. Jin, J. R. Cordy, T. R. Dean, "Where's the Schema? A Taxonomy of Patterns for Software Exchange," in *IWPC*, 2002, pp. 65–74.

[8] A. Fuhr, A. Winter, U. Erdmenger, T. Horn, U. Kaiser, V. Riediger, W. Teppe, "Model-Driven Software-Migration - Process Model, Tool Support and Application," in *Migrating Legacy Applications: Challenges in Service Oriented Architecture and Cloud Computing Environment*, Hershey, PA: IGI Global, 2012.

[9] T. Horn, J. Ebert, "The GReTL Transformation Language," in *Theory and Practice of Model Transformations - 4th International Conference, ICMT 2011.* Zurich, Switzerland: Springer, Jun. 2011, pp. 183–197.

[10] J. Ebert, D. Bilderhauer, V. Riediger, and T. Horn. (2012) Graphenlabor (GRALAB). http://www.uni-koblenz-landau.de/koblenz/fb4/institute/IST/AGEbert/MainResearch/Graphentechnologie/GraLab.

[11] M. Schröder, "Erfassung des Energieverbrauchs von Android Apps," Diploma thesis, Carl von Ossietzky University Oldenburg, 2013.

[12] Robotmafia.org. (2012) GPS Print. https://play.google.com/store/apps/details?id=com.tyfon.gpsprint&hl=en.