TOWARDS ENERGY-AWARE CODING PRACTICES FOR ANDROID

João SARAIVA*, Marco COUTO*, Csaba SZABÓ**, Dávid NOVÁK**
*HASLab/INESC TEC, University of Minho, Portugal,
E-mail: marco.l.couto@inesctec.pt, saraiva@di.uminho.pt
**Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic, Tel.: +421 55 602 4120,
E-mail: csaba.szabo@tuke.sk, david.novak.2@student.tuke.sk

ABSTRACT

This paper studies how the use of different coding practices when developing Android applications influence energy consumption. We consider two common Java/Android programming practices, namely string operations and (non) cached image loading, and we show the energy profile of different coding practices for doing them. With string operations, we compare the performance of the usage of the standard String class to the usage of the StringBuilder class, while with our second practice we evaluate the benefits of image caching with asynchronous loading. We externally measure energy consumption of the example applications using the Trepn profiler application by Qualcomm. Our preliminary results show that selected coding practices do significantly affect energy consumption, in the particular cases of our practice selection, this difference varies between 20% and 50%.

Keywords: *green computing, Android software optimization, code practices, energy consumption of software*

1. INTRODUCTION

The widespread use of mobile devices and of cloud computing to store data are making energy consumption one of the main concerns for software developers. In fact, computer/software execution time is no longer the main concern: Energy is becoming an increasing bottleneck [17]. Most recent approaches to reduce energy consumption focus on the hardware aspect of computers. This is the natural approach: it is the hardware which literally consumes energy. However, very much like a driver the car's fuel consumption, the software can also drastically influence the energy consumed by the hardware!

Unfortunately, there is no software engineering discipline providing techniques nor tools to help software developers to analyze/optimize the energy consumption of their software! As a consequence, software engineers lack guidance as to how to improve the energy consumption of their software systems and which programming practices are most useful. The software engineering community has realized this problem, and recently there are several works showing the programmers concerns about energy consumption [17, 21], the energy savings suggested by official Android coding practices [8], the energy consumption of software testing [10], the energy consumption per source code line [9], the use of models [13, 23], importance of the academia [26], etc.

In this paper, we study the influence in energy consumption of two programming practices in the context of Android software development: a widely used software ecosystem to develop mobile Java-based software applications. In such a setting energy consumption is a main concern. The coding practices we study include one that usually occurs in many Java software system, namely the use of object strings and their operations, and a second one that is relevant in a mobile setting: the use (or not) of caching when loading images from the Internet. Practices, that are often not considered as important by discrete systems [24] and, in general by formal methods of software development [22, 25]. In this paper, we study in detail how different implementations of these two programming problems may influence the overall energy consumption of the software. Furthermore, we describe techniques to monitor the energy consumption of Android applications, the monitoring of the energy consumption when such software is running. Our results show that coding practices do have a great impact in the energy consumption of Android applications. In both cases we analyzed one of the coding solutions was always more energy efficient than the other. As a consequence, Android developers should be aware of such practices so that they are able to develop greener applications.

This paper is organized as follows: Section 2 describes the Android energy monitoring framework we will use to measure the energy consumption of Android programs. We show how to instrument the applications' source code instrumentation so that energy consumption is measured at runtime. In Section 3 we present the coding practices, and the energy consumption profiles of the different solutions. In Section 4 we discuss related work, and in Section 5 we present our conclusions.

2. ANDROID ENERGY MONITORING FRAMEWORK

Mainly two measuring techniques will be used to monitor the energy consumption of Android applications: *internal* and *external* measurements. Both methods are able to provide reliable results however each method is targeted at a different expected type of results.

Internal measuring will be provided by the *TimingLogger class* bundled by default with the *Android Studio version 1.4 package*. This class measures internal code execution time providing the ability to create breakpoints along the way of the code execution. An example showing the use of this class is presented in the listing of Fig. 1. Firstly, we need to import the

android.util.TimingLogger class. The usage for measuring purposes follows the following logic: create a new instance of the TimingLogger class with two logging String parameters, make a call to the *.addSplit()* method (at least one call is required for getting results) after each block of code that is required to be measured and finally call the *.dumpToLog()* method to print out the results into the logs. Notice the placement of the code block to be measured in the *onResume()* method – it is a good practice not to execute any code in the *onCreate()* method that is not related to application initialization otherwise the main thread may be slowed down by executing unnecessary code and the application will be forced into the *Application Not Responding state*.

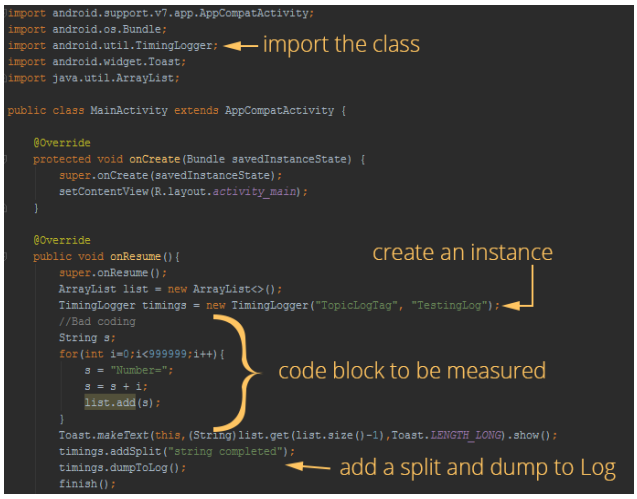


Fig. 1 Basic structure shown in Android Studio

External measuring will be provided by the *Trepn profiler application* developed by Qualcomm: a popular hardware manufacturer for mobile devices[1]. Providing the ability to profile the whole running system or a standalone application it is an ideal tool for the job. The application provides a choice to profile the whole system or only a single application at a given time.

When profiling a single application there is a choice in the Settings section to choose which system parameters should be profiled. The results of the profiling run can be saved as a .csv file which is a set of comma separated values or a .db file which is a database version of the results. The database version is much more versatile and easily analysable in the application itself.

2.1. Device prerequisites

To obtain the most accurate results when profiling the energy consumption of an Android application, every entity that may contribute to additional unwanted power consumption needs to be disabled or stopped. The CPU or GPU load for each running service or application (even in the background) may also influence the energy consumption. Thus, only the necessary core services should be running when doing the energy measurements.

3. ENERGY EFFICIENCY OF TWO ANDROID CODING PRACTICES

Android applications are typically written in the Java Object Oriented programming language. Java programs rely on the Java Virtual Machine (JVM) to execute (bytecode) programs. Android is not different: it uses the Android Runtime (ART) virtual machine. Despite these differences basic Java coding guidelines can have a big impact on performance of an Android application.

3.1. Java-based String Operations

Strings operations are widely used in all programming languages. The Java language provides two ways of defining strings and implementing usual operations, like string insertion and concatenation, namely the use of the standard string operations or by using the *StringBuilder class*. The use of standard string operations, however, may have a significant impact in the overall energy consumption of an Android app.

Consider for example that we wish to define an *ArrayList*, iterate 999999 times whilst filling the list with Strings in format 'Number=1' and adding the newly created String to an *ArrayList*, finally print out the last String in the ArrayList.

This simple problem can be expressed via standard string operations as follows:

```
String s = "";
for(int i=0;i<999999;i++){
      s = "Number=";
      s = s + i;
      list.add(s);
}
```

The alternative relies on the *StringBuilder class* defined by default in Java instead of creating a new String in each iteration, as shown next.

```
final String txt = "Number=";
StringBuilder sb = new StringBuilder();
for(int i=0;i < 999999;i++){
      sb.setLength(0);
      sb.append(txt);
      sb.append(i);
      list.add(sb.toString());
}
```

To have significant results many iterations are needed to cut down on errors or values that are significantly out of ordinary bounds. The number of iterations – in this example 999999 – helps to exaggerate the time difference between code execution times of the first and second approaches. To have reliable results we executed the two programs 100 times. Both programs were executed in a LG G3 smartphone, running Android 5.0, with a Qualcomm Snapdragon 801 CPU. Due to space limitations we only show the 5 most significant in Tab. 1 (with the biggest deviation from the average value).

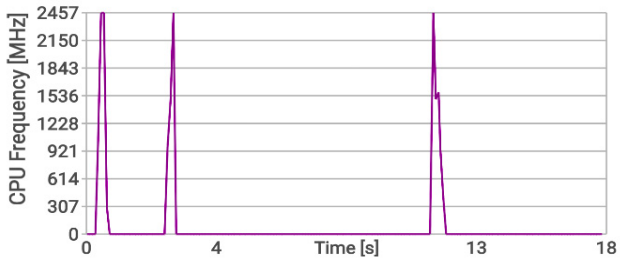
[1] https://www.qualcomm.com/

Table 1 Internal code execution time

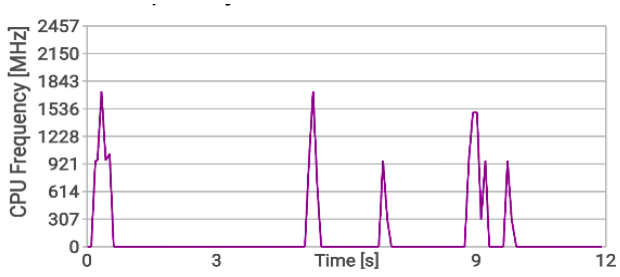
Run #	Approach #1 [ms]	Approach #2 [ms]
1	2936	1693
2	4320	2184
3	4496	2478
4	3285	2374
5	4025	2149
Average	3812.4	2175.6

The results show that the *StringBuilder* approach runs 1636,8ms faster than the standard string operations one, even when performing a simple operation.

Fig. 2 visually displays the CPU frequency of the Qualcomm CPU when running the two programs, as provided by the Trepn profiler.



a) Standard *String* Operations



b) *StringBuilder* Operations

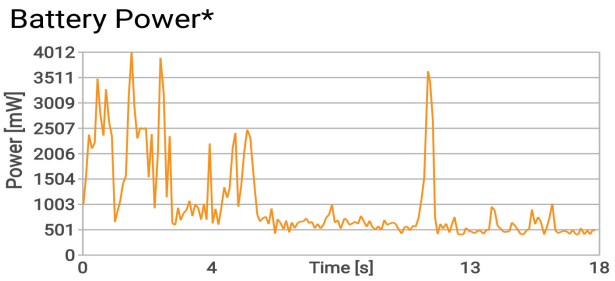
Fig. 2 *Trepn* profiler graph for a) Standard String Operations and b) StringBuilder Operations

As it can be observed from the Trepn graphics the use of standard string operations in Java is much more CPU intensive than the StringBuilder method, which not only justifies its longer running time, but also its higher energy consumption as discussed next.

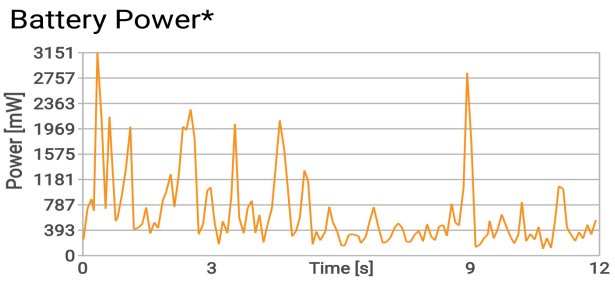
In Fig. 3 we show the Trepn graphics with the batery usage when running both programs, where we clearly see higher energy peaks when performing standard string operations. For the standard string operations it peaks at 4012mW whereas for the second approach it peaks at only 3157mW which is a saving of 21,31%.

3.2. Image Caching on Loading

Image loading in Android can be performed in both synchronous or asynchronous way. Doing it synchronously implies that the application waits until the images are fully loaded. On the other hand, with asynchronous loading the application can still work normally while without waiting for images to load. It focuses on loading lighter content (text) before loading heavy content (images and bitmap graphics).



a) Standard *String* Operations



b) *StringBuilder* Operations

Fig. 3 *Trepn* profiler graph for battery use for a) Standard String Operations and b) StringBuilder Operations

Light content is loaded instantaneously and the heavier content is loaded asynchronously, i.e. in the background in another thread and displayed once loaded hence the main thread is not blocked by this load process. In the meantime a placeholder image is displayed instead of a blank space. Fig. 4 shows an example of this techniques being used.

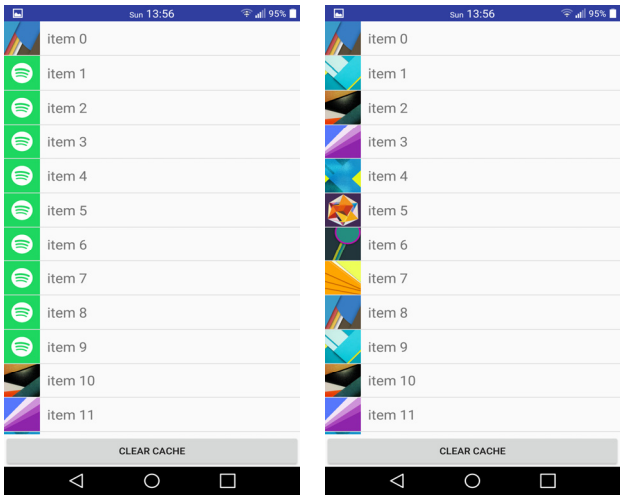
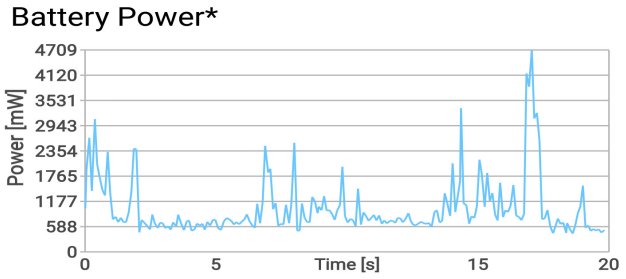


Fig. 4 Lazy loading images with placeholders

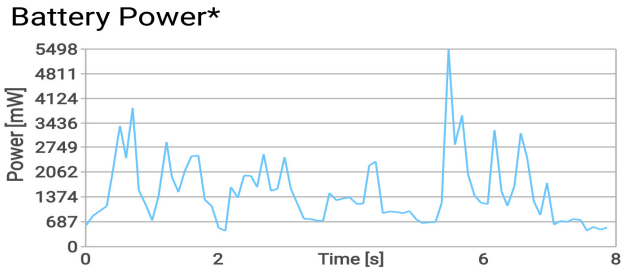
Although asynchronous loading already improves the performance and responsiveness of the application, the efficiency can be improved further by caching the images. Results are beneficial for the user and the performance as well. The only condition is that the application has the permissions to write to the storage either external or internal and there is enough space available to save the

images. Caching reduces data traffic as there are no redundant downloads and it is faster to load an image from the device storage than downloading it over and over again.

In order to test the influence of caching images in energy consumption, we developed two versions of the same application (represented in Fig. 4): with and without image caching enabled. We then tested both of them and tracked the energy consumption, the network traffic and the memory usage in both cases. The energy consumption results can be seen in Fig. 5.



a) No image caching



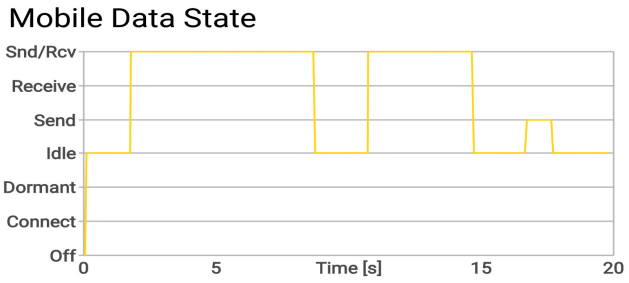
b) Image caching

Fig. 5 Comparing a) no image caching vs. b) image caching concerning energy consumption

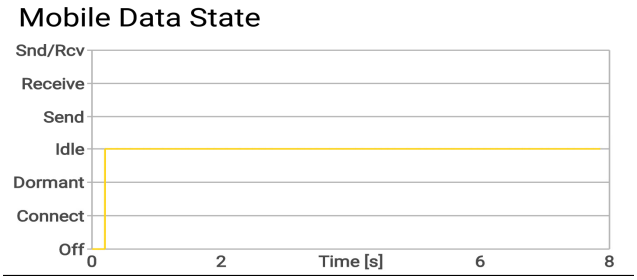
As we can observe in Fig. 5, when loading the same image, the cache-based approach is 60% faster than the non-cached one. Fig. 6 displays the battery usage of both solutions. It is clear from this figure that the cache-based solution contains a higher energy peak: 5498mW with image caching, and 4709mW with no image caching. However, because the cached approach is faster performing the task, it is approximately 50% more energy efficient than the non-cached solution.

Another interesting aspect of our analysis was the network traffic consumed in both approaches. After the initial download of all images no unnecessary network traffic occurs with the caching enabled, whereas with the caching disabled the device initialized three network transactions to download images, as shown in Fig. 6. Such behavior resulted in higher battery consumption and overall higher power consumption peaks (as seen on Fig. 5).

Concerning memory usage, we can observe by analyzing Fig. 7 that enabling image caching does not have a significant influence in the application's memory usage. The graphs for the approach with/without caching approach are very similar, with very close values for highest and lowest peak.

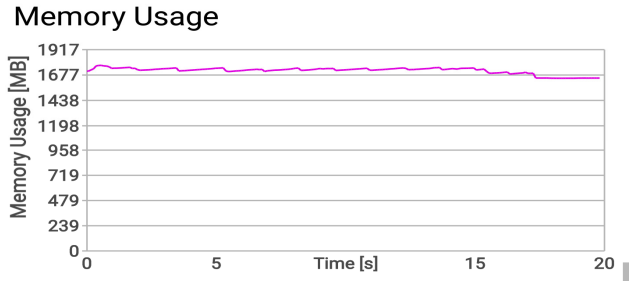


a) No image caching

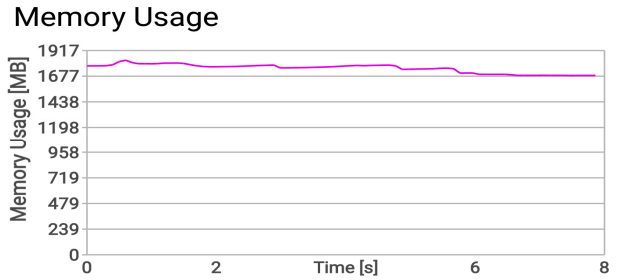


b) Image caching

Fig. 6 Comparing a) no image caching vs b) image caching concerning network traffic



a) No image caching



b) Image caching

Fig. 7 Comparing a) no image caching vs b) image caching concerning memory usage

4. RELATED WORK

Energy consumption awareness has brought up an increasing interest in analyzing the energy efficiency of software systems. Developers seem to be now more focused on reducing energy consumption through software improvement [17], since it is the software that triggers the hardware behavior. This principle guided several research works that appeared in the last decade.

Studies have shown that the energy consumption of a software system can be significantly influenced by a lot of factors. Such factors can be software development related, such as code obfuscation [20], different design patterns or coding practices [1, 8, 11, 18], refactorings [19], and even the usage of different data structures [12, 16]. Even in software testing the decisions made can influence the overall energy consumption at the testing phase [10].

With the propagation and mass usage of mobile devices, the energy consumption issue became even more relevant. This is due to the fact that mobile devices run over batteries with limited capacity. Motivated by this, several research works focused on providing information about the energy consumption of applications. Some of them were able to monitor the consumption per application [4, 7, 27, 28], or even compare different usages of similar applications [6], while others tried to determine the energy consumed by blocks of code, such as methods [2, 3], lines of code [9] or even bytecode instructions [5].

Most of the works in this are based on energy models: a prediction model able to determine the energy consumption by matching information retrieved from hardware components to previously determined measurements [13, 14, 15, 28].

5. CONCLUSIONS

In this paper we discussed two common Android coding practices and how such practices influence energy consumption. We showed that the use of the *StringBuilder class* to manipulate strings and the use of a cache when loading images from internet are more energy efficient than the use of standard string operations and non-cache based approaches. We have presented examples of equivalent Java programs that use (not use) standard strings (cache images), and we have measured the energy consumption of both solutions when executed on a smartphone running Android. Our preliminary results show a maximum energy saving of 50%.

To generalize our results, we proved a part of energy saving practices to be adequate to their aim. But, what is more important, these results imply that the code base can be optimized to save energy of working software running on Android but other platforms as well.

The ultimate goal of identifying energy (in)efficient coding practices is a first step towards developing a full catalog of *coding practices* for guiding software engineers in developing energy-aware software. Thus, as future research we plan to not only extend our coding practices, with other coding practices found in literature, namely,

- the use of coding practices that are proposed in the official Android developers web site: network usage, memory consumption, and low-level programming practices that were studied in [8],
- the use of the most energy efficient collections that are part of Java Collections Framework library [12, 16].

The newest version of the Java programming language includes *Streams* that provide a powerful mechanism to define methods that traverse data structures. Streams provide a coding practice widely used by functional programmers. As future work we plan to study how such (functional) coding practices influence energy consumption in Android.

ACKNOWLEDGMENT

This work is funded by the Slovak Research and Development Agency under the contract No. SK-PT-2015-0037 and by the Portugal-Slovakia Cooperation FCT Project (Ref. 441), and by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT – Fundacão para a Ciência e a Tecnologia within project POCI-01-0145-FEDER-016718.

REFERENCES

[1] BRANDOLESE, C. – FORNACIARI, W. – SALICE, F. – SCIUTO, D.: The impact of source code transformations on software power and energy consumption, *Journal of Circuits, Systems, and Computers*, Vol. 11, No. 5, pp. 477–502, 2002.

[2] COUTO, M. – CUNHA, J. – FERNANDES, J. P. – PEREIRA, R. – SARAIVA, J.: Greendroid: A tool for analysing power consumption in the android ecosystem, in *Scientific Conference on Informatics, 2015 IEEE 13th International*, Nov 2015, pp. 73–78.

[3] COUTO, M. – TIAGO, C. – CUNHA, J. – FERNANDES, J. P. – SARAIVA, J.: Detecting anomalous energy consumption in android applications, in *Programming Languages, ser. Lecture Notes in Computer Science*, F. M. Quintão Pereira, Ed. Springer International Publishing, 2014, Vol. 8771, pp. 77–91.

[4] FLINN, J. – SATYANARAYANAN, M.: Powerscope: A tool for profiling the energy usage of mobile applications, in *Proc. of the Second IEEE Workshop on Mobile Computer Systems and Applications*, ser. WMCSA'99, IEEE Computer Society, 1999.

[5] HAO, S. – LI, D. – HALFOND, W. – GOVINDAN, R.: Estimating android applications' cpu energy usage via bytecode profiling, in *Green and Sustainable Software (GREENS), 2012 First International Workshop on*, June 2012, pp. 1–7.

[6] JABBARVAND, R. – SADEGHI, A. – GARCIA, J. – MALEK, S. – AMMANN, P.: Ecodroid: An approach for energy-based ranking of android apps, in *Proc. of the Fourth International Workshop on Green and Sustainable Software*, ser. GREENS '15, IEEE Press, 2015, pp. 8–14.

[7] KJÆRGAARD, M. – BLUNCK, H.: Unsupervised power profiling for mobile devices, in *Mobile and Ubiquitous Systems: Computing, Networking, and Services, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, A. Puiatti and T.

Gu, Eds. Springer Berlin Heidelberg, 2012, Vol. 104, pp. 138–149.

[8] LI, D. – HALFOND, W. G. J.: An investigation into energy-saving programming practices for android smartphone app development, in *Proc. of the 3rd International Workshop on Green and Sustainable Software*, ser. GREENS 2014, ACM, 2014, pp. 46–53.

[9] LI, D. – HAO, S. – HALFOND, W. G. J. – GOVINDAN, R.: Calculating source line level energy information for android applications, in *Proc. of the 2013 International Symposium on Software Testing and Analysis*, ser. ISSTA 2013, ACM, 2013, pp. 78–89.

[10] LI, D. – JIN, Y. – SAHIN, C. – CLAUSE, J. – HALFOND, W. G. J.: Integrated energy-directed test suite optimization, in *Proc. of the 2014 International Symposium on Software Testing and Analysis*, ser. ISSTA 2014, ACM, 2014, pp. 339–350.

[11] LINARES-VÁSQUEZ, M. – BAVOTA, G. – BERNAL-CÁRDENAS, C. – OLIVETO, R. – DI PENTA, M. – POSHYVANYK, D.: Mining energy-greedy api usage patterns in android apps: An empirical study, in *Proc. of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014, ACM, 2014, pp. 2–11.

[12] LIU, K. – PINTO, G. – LIU, Y. D.: Data-oriented characterization of application-level energy optimization, in *Fundamental Approaches to Software Engineering, ser. Lecture Notes in Computer Science*, A. Egyed and I. Schaefer, Eds., Springer Berlin Heidelberg, 2015, Vol. 9033, pp. 316–331.

[13] NAKAJIMA, S.: Model-based power consumption analysis of smartphone applications, in *16th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2013)*, Miami, Florida, USA, September 29th, 2013.

[14] NAKAJIMA, S.: Model checking of energy consumption behavior, in *Complex Systems Design & Management Asia*, M.-A. Cardin, D. Krob, P. C. Lui, Y. H. Tan, and K. Wood, Eds. Springer International Publishing, 2015, pp. 3–14.

[15] PATHAK, A. – HU, Y. C. – ZHANG, M. – BAHL, P. – WANG, Y.-M.: Fine-grained power modeling for smartphones using system call tracing, in *Proc. of the Sixth Conference on Computer Systems, ser. EuroSys '11,* ACM, 2011, pp. 153–168.

[16] PEREIRA, R. – COUTO, M. – SARAIVA, J. – CUNHA, J. – FERNANDES, J. P.: The influence of the java collection framework on overall energy consumption, in *Proc. of the 5th International Workshop on Green and Sustainable Software*, ser. GREENS '16, ACM, 2016, pp. 15–21.

[17] PINTO, G. – CASTOR, F. – LIU, Y. D.: Mining questions about software energy consumption, in *Proc. of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014, ACM, 2014, pp. 22–31.

[18] SAHIN, C. – CAYCI, F. – GUTIERREZ, I. L. M. – CLAUSE, J. – KIAMILEV, F. – POLLOCK, L. – WINBLADH, K.: Initial explorations on design pattern energy usage, in *Green and Sustainable Software (GREENS), 2012 First International Workshop on*. IEEE, 2012, pp. 55–61.

[19] SAHIN, C. – POLLOCK, L. – CLAUSE, J.: How do code refactorings affect energy usage?, in *Proc. of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '14, ACM, 2014, pp. 36:1–36:10.

[20] SAHIN, C. – TORNQUIST, P. – MCKENNA, R. – PEARSON, Z. – CLAUSE, J.: How does code obfuscation impact energy usage?, in *Proc. of the 2014 IEEE International Conference on Software Maintenance and Evolution*, ser. ICSME '14, IEEE Computer Society, 2014, pp. 131–140.

[21] SANTOS, M. – SARAIVA, J. – PORKOLÁB, Z. – KRUPP, D.: Energy Consumption Measurement of C/C++ Programs Using Clang Tooling, *Z. Budimac (ed.): Proceedings of the SQAMIA 2017: 6th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications*, Belgrade, Serbia, 11-13.9.2017, Paper No. 15, 8 pages, Also published online by CEUR Workshop Proceedings No. 1938 (http://ceur-ws.org) ISSN 1613-0073.

[22] ŠIMOŇÁK, S.: Formal methods transformation optimizations within the ACP2PETRI tool, *Acta Electrotechnica et Informatica,* Vol. 6, No. 1, 2006, pp. 75–80, ISSN 1335-8243.

[23] ŠIMOŇÁK, S. – HUDÁK, Š. – KOREČKO, Š.: APC semantics for Petri nets, *Informatica*, Vol. 32, No. 3, 2008, pp. 253–260, ISSN 0350-5596.

[24] ŠIMOŇÁK, S. – PEŤKO, I.: Patool – a tool for design and analysis of discrete systems using process algebras with FDT integration support, *Acta Electrotechnica et Informatica,* Vol. 10, No. 1, 2010, pp. 59–67, ISSN 1335-8243.

[25] ŠIMOŇÁK, S. – ŠOLC, M.: Enhancing Formal Methods Integration with ACP2Petri, *Journal of Information and Organizational Sciences,* Vol. 40, No. 2 (2016), pp. 221–235, ISSN 1846-9418.

[26] SZABÓ, Cs. – SARAIVA, J.: Focusing software engineering education on green application development, *Conference of Information Technology and Development of Education - ITRO 2017*, Novy Sad, Serbia, pp. 165–169, ISBN 978-86-7672-302-7.

[27] YOON, C. – KIM, D. – JUNG, W. – KANG, C. – CHA, H.: Appscope: Application energy metering framework for android smartphone using kernel activity monitoring, in Presented as part of the *2012 USENIX Annual Technical Conference (USENIX ATC 12),* USENIX, 2012, pp. 387–400.

[28] ZHANG, L. – TIWANA, B. – QIAN, Z. – WANG, Z. – DICK, R. P. – MAO, Z. M. – YANG, L.: Accurate online power estimation and automatic battery behavior based power model generation for smartphones, in *Proc. of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, ser. CODES/ISSS '10, ACM, 2010, pp. 105–114.

Received October 25, 2017, accepted March 8, 2018

BIOGRAPHIES

João Saraiva is an Auxiliar Professor at the Departmento de Informática, Universidade do Minho, Braga, Portugal, and a researcher member of HASLab/INESC TEC.He obtained a MSc degree from University do Minho in 1993 and a Ph.D. degree in Computer Science from Utrecht University in 1999. His main research contributions have been in the field of programming languages design and implementation, program analysis and transformation, and functional programming. In the last 4 years he turned his attention to energy consumption analysis in software, and managed to obtain already several accepted papers in the area, and a few research projects. He supervised 4 PostDoc projects, 7 PhD projects (5 awarded and 3 running) and over 20 (Pos-Bologna) MSc thesis. He counts with over 60 international publications, he has served in over 50 program committees of international events, and in the evaluation committees of 5 research agencies: ANII (Uruguay), FRS-FNRS (Belgium), NWO (The Netherlands), FWF (Austria), and FCT (Portugal).

Marco Couto completed his MsC degree in Informatics Engineering in 2014, with a thesis entitled "Monitoring Energy Consumption in Android Applications", with a scholarship in a project called GreenSSCM – Green Software for Space Control Missions, at the University of Minho. He is also one of the members and co-founder of the Green Software Lab, at University of Minho. Currently, he is working on his PhD, being a student in the MAP-i doctoral program, and is still working on the energy consumption analysis and energy-aware software areas. In the last years, he managed to publish several of his energy analysis related works, and has been involved in several research projects, financed from several institutions such as FCT (Portugal) and FLAD-NSF (Portugal/United States of America).

Csaba Szabó graduated (MSc.) with distinction at the Dept. of Computers and Informatics in 2003 and obtained his PhD. in Program- and Information Systems at the FEEaI at Technical University of Košice in 2007. Since 2006 he is affiliated as assistant professor with the Dept. of Computers and Informatics. Currently he is involved in research in the field of behavioral description of software, software and test evolution, and testing and evaluation of software. In the last years, he has been involved in different positions of research projects financed by different agencies such as APVV (Slovakia-Austria, Slovakia-Portugal), KEGA, VEGA (both Slovakia).

Dávid Novák joined the research on green software in 2015, in the second year of his Bachelors' studies. He defended his final thesis entitled "Green Software Development for the Android Platform" at the department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at Technical University of Košice in June 2016. He was a member of the joint APVV/FCT project Slovakia-Portugal. He is working as frontend/backend developer at Hellephant company.