10th International Conference on Global Software Engineering

www.icgse.org

13-16 July 2015, Ciudad Real, Spain

2006- 2015: 10 years of ICGSE

ICGSE

ICGSEW

2015 IEEE 10th International Conference on Global Software Engineering Workshops

A first approach on legacy system energy consumption measurement

Victor Cordero, Ignacio García-Rodríguez de Guzmán, Mario Piattini
Institute of Information Technologies and Systems
University of Castilla-La Mancha
Ciudad Real, Spain
Victor.Cordero@alu.uclm.es, {Ignacio.GRodriguez, Mario.Piattini}@uclm.es

***Abstract*—Nowadays, software sustainability is growing in importance. Not only IT infrastructure is becoming greener, but also software. It is possible to find methods and methodologies intended to produce more sustainable software with lower power consumption. In spite the slow evolution of software engineering towards " Green software", there exist a huge amount of legacy systems still running in organizations. Is then necessary to develop such systems from scratch in order to make them more sustainable? Probably, the most logical and appropriate answer for this question is no, since existing software can be refactored in order to improve its *greenability* quality characteristic. As a first step towards power consumption improvement, the authors propose a tool to analyze legacy systems in order to detect parts of the system with higher energy consumption. Using the *profiling* technique, the proposed tool instrument legacy Java systems in order to keep track of its execution. This information, together with the energy consumption (logged by means a data logger hardware), enables the engineer to analyze legacy system consumption detecting energy peaks in the system (e.g. the PC). The analysis gives the engineer evidences about candidates to be refactored in order to reduce energy consumption.**

Keywords—software energy consumption, energy consumption, profiling, software sustainability, energy data logger, reverse engineering

I. Introduction

The interest, necessity and efforts on designing and implementing green IT solutions has been increasing dramatically over the past few years [1]. One of the most influencing reasons which drive this green trend is the IT's rapidly rising energy demands, due to growing global adoption of computing services [2].

New models and proposals are appearing in the quest to produce sustainable software [3], but all these approaches do not take into account the huge amount of Information Systems still running in all the organizations. Therefore is very likely that these software systems would not be sensitive about any kind of impact on the environment and are thus not by considered as green IT.

In the road to improve IT's greenability Legacy Information Systems cannot be discarded, since from a functionality point of view such systems do not present any problem. Other problem is to determine what does it mean to improve the sustainability quality characteristic of a software system. In a few words, such improvement could be understood as carrying out refactoring tasks in order to improve the system quality (e.g. sustainability) without changing the functionality. Assuming this consideration, should the refactorization be applied over the whole legacy system or must be directed to these parts (e.g. methods, modules, loops, hard-drive accesses, etc.)? In response to that question it is important to highlight two important challenges:

- Refactorization is a time- and effort-consuming task, so it must be only applied against these parts of the legacy system that causes excessive power consumption. In these respect,
- What does really cause high power consumption in a software system?

Whit all this ideas in mind, the authors of this paper proposes GreeSoM (**Gree**n **So**ftware energy **M**easurement), a tool intended to measure software energy consumption in order to detect consumption peaks and work out a set of candidate software structures involved in such consumption peaks. GreeSoM is not a tool focused on obtaining an accurate value of software energy consumption, but information of consumption peaks merged with detailed execution information of the legacy system. It is possible to obtain candidate lists of software structures that would be involved in consumption peaks by merging software energy consumption records with software execution traces. With such information, the engineer has the chance to analyze such particular structures, methods or functions and improve them making in turn the legacy system more sustainable by decreasing the energy consumption. So GreeSoM plays two important roles: (i) detect consumption peaks and associate the fragments of the legacy system responsible of such consumption; and (ii) serve as a analysis platform in order to find situations, patterns or structures that repetitively causes abnormal power consumption in software systems. While the first role of GreeSoM is intended to improve the sustainability quality characteristic of software systems, the second is focused on building a "green software engineering knowledge" related with software sustainability by analysing software system and extracting conclusions of the case studies.

So, summarizing, GreeSoM is a tool intended to carry out a white-box energy consumption analysis to Java based systems. GreeSoM performs such analysis by instrumenting the Java legacy systems (to obtain execution logs) and executing the new version of the code. GreeSoM matches the execution and energy consumption information (captured by an external hardware) in order to detect energy consumption peaks and find out parts of the legacy system responsible of excessive energy consumption.

It is important to point out that GreeSoM has not the ability to measure software energy consumption. This task is carried out by an external hardware data logger, which is connected to the execution environment (PC) and generates log files with information about power consumption.

The remainder of this chapter is organized as follows: section II analyzes briefly some proposals related with software energy consumption; section III presents a detailed explanation of the characteristics of the tool; section IV includes a casa study where the proposed tool is the subject of the case study; finally, section V depict some conclusions and future lines of the presented work.

II. State of the art

On one hand, the most relevant proposals regarding energy consumption measurement proposals will be discussed in this section. On the other hand, the hardware data logger used to keep track of the energy consumption in the proposed tool will be also presented in order to enable the reader with a basic knowledge about the hardware used to measure energy consumption in the system where the legacy system is running.

A. *Software energy consumption analysis tools*

Despite of the fact that exist many proposals for energy measurement [4], most of them are focused on IT infrastructures (servers, network, hardware components, etc.). Nonetheless, it is possible to find a few proposals regarding software power consumption.

One of this proposals is Jalen [5], a software tool that uses the profiling technique and energy models. The energy consumption values obtained by Jalen are calculated by means of an API called *POWER API.* This API obtains an estimation of software energy consumption.

The second proposal is depicts a software tool called Green Tracker [6]. In order to measure the energy consumption attached to given software, this tool estimate the usage that this software makes of the CPU. Green Tracker executes initial benchmarks tasks in order to estimate whether a given system is consuming more energy that expected in this system.

Both proposals implement an estimation of energy consumption. The values obtained by Jalen are calcutalated by means of mathematical models. In the case of Green Tracker, the consumption estimation regarding the software system is obtained by measuring the power consumption of the CPU. For such reason, Green Tracker requires also an external power consumption logger to keep track of energy consumption of the system.

Another important approach is the one presented in [11], where the authors present metrics for evaluating energy efficiency in software, as well as an approach to use such metrics through software development cycle. Also, the authors propose the code instrumentation as a key point in order to obtain detailed information about software consumption not only about the whole system, but also about the different modules that implement it.

B. *Energy consumption hardware*

GreeSoM implements the algorithms and logic in order to evaluate the software energy consumption, nonetheless, the energy consumption data must be provided by an external hardware. In the presented approach, the measurement hardware chosen is the multipurpose measurement *HOBO UX120 Data Logger* with an additional transducer sensor to measure amperes.

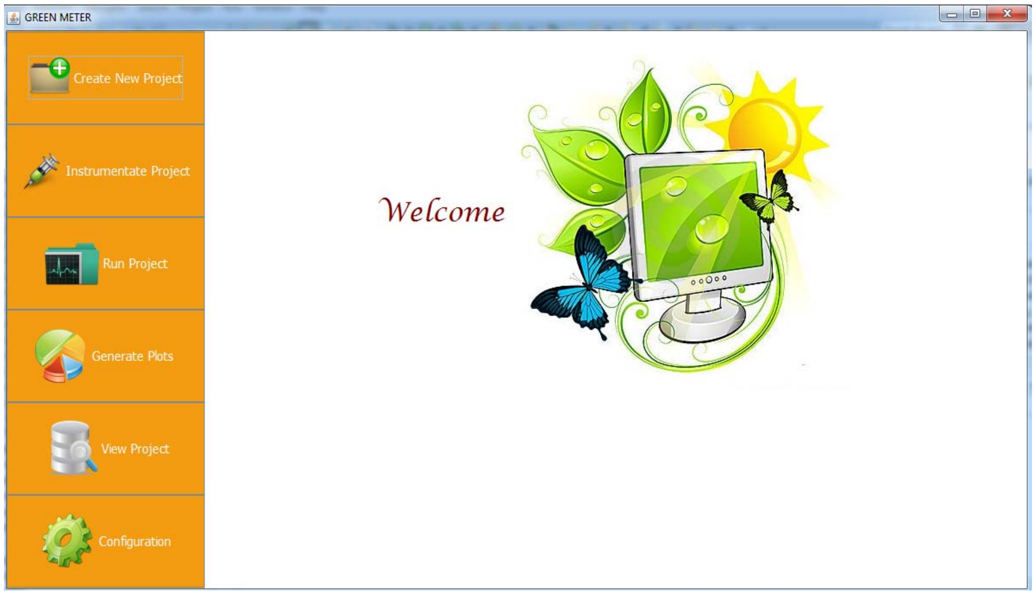


Fig. 1. Main interface of the support tool

III. MEASURING POWER CONSUMPTION IN LEGACY SYSTEMS:THE TOOL

In this section GreeSoM, the tool supporting the consumption measurement process will be presented. GreeSoM (see **Fig. 1**) implements the following functionalities:

- Project driven: The engineer could create a consumption measurement analysis. For each new project, the tool creates a copy of the legacy system, so the original one keeps unmodified.
- Instrumentation: Depending of the availability of the legacy system (source code or bytecode), the tool generates an instrumented copy of the legacy system. This process will be further explained. Instrumentation injects sentences into a copy of the legacy system in order to product log files with execution traces.
- Legacy System Execution: GreeSoM can launch and stop the execution of an instrumented version of the legacy system. This functionality is completely driven by the user, since the user is the responsible of deciding with usage scenarios must be executed. It is highly recommended to plan the usage scenarios of the system in order to obtain a more structured and relevant information in the following analysis.
- Data Analysis: The execution and consumption information is merged in order to correlate the system (PC) consumption with the legacy system execution. The energy consumption-measuring unit (produced by the energy measurement hardware) is the *ampere*.

Next, the instrumentation execution of the legacy system and data analysis will be explained in detail.

A. Instrumentation of the Legacy System

Firstly, the legacy system must be prepared to enable the generation of execution traces. It is important to point out the different possibilities that GreeSoM can deal with in the instrumentation stage of the legacy system:

1. Java Source code is available. Source code is available (the owning organization hast the legacy system source code AND can provide it). GreeSoM implements a Java *parser* that analyzes the code and instrument the code with specific sentences in order to generate *log* files with specific execution information such as. The Java parser implemented by the tool analysis the source code and produce a functional equivalent version of the legacy system with a new (but harmless) functionality: the ability to generate log files recording the execution information of the legacy system. Each execution record is made up of: execution thread, executing class, time and method. An execution record will be generated only under two circumstances: (i) a method is starting or (ii) finishing its execution. Once the instrumented version of the system is ready, it is only required to compile it. At the moment, GreeSoM only deals with Java-based systems, since the instrumentation state is highly platform-dependent.
2. Only Java bytecode is available. Sometimes, source code is not available due some reasons: (i) legacy system is so legacy that the source code is lost; (ii) the legacy system has been developed by a third party, so the company only owns the binary files; or (iii) legal restrictions does not allow a third party to examine the source code. So, in such circumstances, it is not possible include sentences

to produce execution information (at least, Java sentences). Therefore, the "*.class*" files must be instrumented. For such an end, the ASM [7] platform has been used. In just a few words, ASM is a framework that uses a visitor-based approach to generate new bytecode from an existing one introducing new instructions, but without modifying the current functionalities.

Secondly, the legacy system must now be executed in an ordinary way. This new version of the legacy system will now produce log files containing execution records with the aforementioned information (see **Fig. 2**). The legacy system will be producing execution information without interruption while executing. The execution of the instrumented legacy system is independent of GreeSoM.

B. *Execution of the Legacy System*

In the execution stage, not only the execution information of the legacy system is collected (as log files), but also the consumption information. In this case, the in order to record the electric consumption an external hardware has been used together with GreeSoM: the HOBO UX120 Analog Data Logger. This hardware offer support to make measures different kind of measures (depending on the connected sensors) such as temperature, CO2, or in the context of this work, energy consumption.

The data logger is connected to the PC running the instrumented system, so, the data logger stores consumption records while the system is executing and generation execution traces. The data logger is set up with the same PC that executes the instrumented legacy system, so, both the PC and the data logger shares the same time and date. This is a crucial detail as will be seen in the analysis stage.

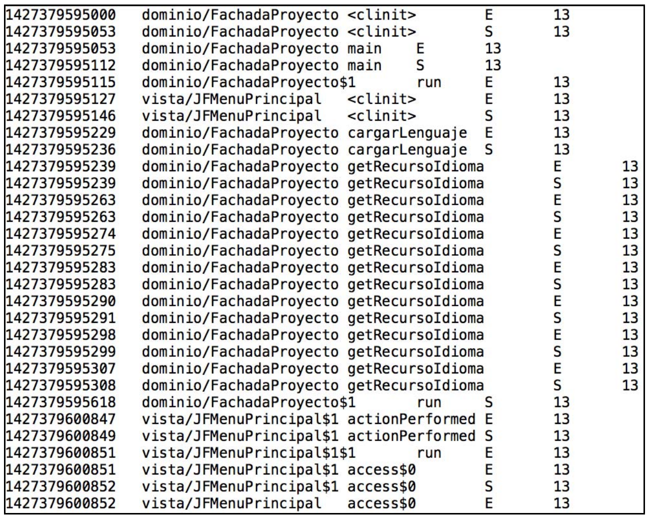
C. *Analysis of the information recorded*

Finally, the next step when the legacy system execution has finished is the analysis of the collected data. On one hand, the instrumented legacy system produces execution log files (see **Fig. 2**) (detailed records with the execution of each method of the legacy system). On the other hand, the data logger stores internally all the consumption records of the PC where the legacy system has been running. The consumption records are represented by means a "*csv*" file (see **Fig. 3**). In this third stage, the analysis of the data is carried out. For such an end, the presented approach takes as input both the execution and the consumption log files.

The first step to carry out the consumption analysis is the merging of the execution and consumption information. The merging process present a sampling frequency problem: execution files keeps a record each time a entry or exit method is executed, but the consumption files store a consumption record each second. When merging the information, for each consumption record it is possible to associate more than one (probably several tens) execution records. This fact establishes a certain degree of uncertainty consisting on "which instruction (or group of instructions) could be in charge of a given consumption peak".

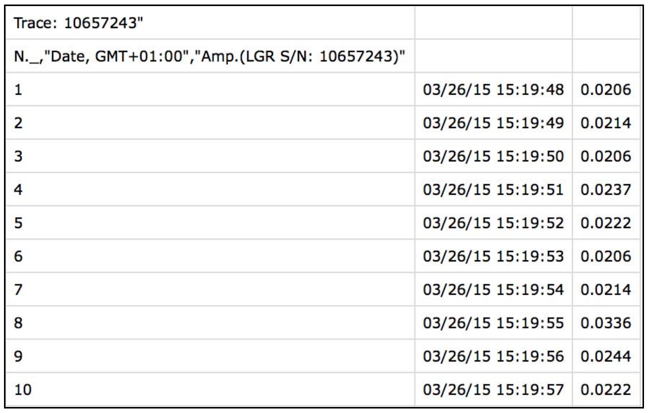
So, from the log files (execution and consumption), our framework proposes two analysis of the information:

- Execution information: By analyzing the records of the execution log, it is possible calculate the time spend in each method, class and package of the legacy systems in relation with the total execution time. With this information in mind, it is possible estimate the relevance or importance of a class or a package regarding the whole legacy system. This aspect is closely related with the technical debt concept [8].



```
1427379595000   dominio/FachadaProyecto <clinit>        E       13
1427379595053   dominio/FachadaProyecto <clinit>        S       13
1427379595053   dominio/FachadaProyecto main    E       13
1427379595112   dominio/FachadaProyecto main    S       13
1427379595115   dominio/FachadaProyecto$1       run     E       13
1427379595127   vista/JFMenuPrincipal   <clinit>        E       13
1427379595146   vista/JFMenuPrincipal   <clinit>        S       13
1427379595229   dominio/FachadaProyecto cargarLenguaje  E       13
1427379595236   dominio/FachadaProyecto cargarLenguaje  S       13
1427379595239   dominio/FachadaProyecto getRecursoIdioma        E       13
1427379595239   dominio/FachadaProyecto getRecursoIdioma        S       13
1427379595263   dominio/FachadaProyecto getRecursoIdioma        E       13
1427379595263   dominio/FachadaProyecto getRecursoIdioma        S       13
1427379595274   dominio/FachadaProyecto getRecursoIdioma        E       13
1427379595275   dominio/FachadaProyecto getRecursoIdioma        S       13
1427379595283   dominio/FachadaProyecto getRecursoIdioma        E       13
1427379595283   dominio/FachadaProyecto getRecursoIdioma        S       13
1427379595290   dominio/FachadaProyecto getRecursoIdioma        E       13
1427379595291   dominio/FachadaProyecto getRecursoIdioma        S       13
1427379595298   dominio/FachadaProyecto getRecursoIdioma        E       13
1427379595299   dominio/FachadaProyecto getRecursoIdioma        S       13
1427379595307   dominio/FachadaProyecto getRecursoIdioma        E       13
1427379595308   dominio/FachadaProyecto getRecursoIdioma        S       13
1427379595618   dominio/FachadaProyecto$1       run     S       13
1427379600847   vista/JFMenuPrincipal$1 actionPerformed E       13
1427379600849   vista/JFMenuPrincipal$1 actionPerformed S       13
1427379600851   vista/JFMenuPrincipal$1$1       run     E       13
1427379600851   vista/JFMenuPrincipal$1 access$0        E       13
1427379600852   vista/JFMenuPrincipal$1 access$0        S       13
1427379600852   vista/JFMenuPrincipal   access$0        E       13
```

Fig. 2. Excerpt of a execution log file generated by the instrumented legacy system



Trace: 10657243"		
N._,"Date, GMT+01:00","Amp.(LGR S/N: 10657243)"		
1	03/26/15 15:19:48	0.0206
2	03/26/15 15:19:49	0.0214
3	03/26/15 15:19:50	0.0206
4	03/26/15 15:19:51	0.0237
5	03/26/15 15:19:52	0.0222
6	03/26/15 15:19:53	0.0206
7	03/26/15 15:19:54	0.0214
8	03/26/15 15:19:55	0.0336
9	03/26/15 15:19:56	0.0244
10	03/26/15 15:19:57	0.0222

Fig. 3. Excerpt of the data logger consumption traces

- Consumption information: By merging the execution records (**Fig. 2**) with the consumption ones (**Fig. 3**), it is possible to draw an energy consumption chart (see **Fig. 4**). This energy consumption char represent the fluctuation of the energy consumption (amperes) along with the execution time. At first sight only consumption evolution is represented, however, the tool has a merged representation of both execution and consumption data. As a consequence, the tool enables the expert to choose the most relevant peaks of the graph in order to analyze what is going on in this period (one second).

In order to fully understand what does the consumption chart means, it is important to highlight some important aspects about the colors used to represent the consumption evolution of the energy consumption chart (see **Fig. 4**). The lines of the consumption chart can be painted in three different colors:

- Green: during this period the legacy system has been running all the time. That is to say, the legacy system execution has not been interrupted. This information is concluded from the information of the execution file (see **Fig. 2**).
- Yellow: during this period the legacy system is sharing the CPU with other processes. This represents that the legacy system had a idle-time where other applications have been using the CPU Therefore, there exist some probability that the consumption is not due to the legacy system. This information is concluded from the execution log file when the records do not cover all the time of the period.
- Red: during this period the legacy system has not been executed, and thus, the recorded consumption must be due to other processes running in the CPU. This information, also concluded from the log file, is concluded when there no record representing the execution of any method of the legacy system. In the example depicted in **Fig. 4** there are no red lines since the legacy system of the example has not been stopped time enough (one second).

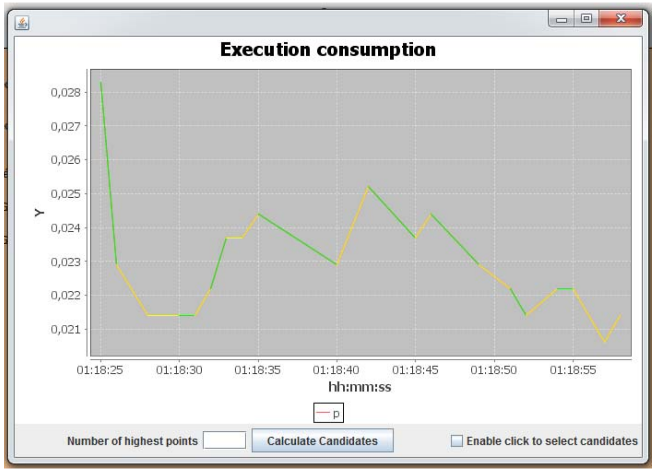


Fig. 4. Example of consumption chart

To analyze consumption peak would be easy, since GreeSoM facilitates the reading of all the methods taking part in this period. Nonetheless, repetitive consumption peaks may reveal that there exist different fragments of the system that cause these peaks. Detecting such fragments of code (methods) would be a very interesting starting point in order to improve software sustainability, but such analysis would become quite complex (in the time of one peak, tens of methods would be executed). In order to provide the engineer with an automated solution to analyze sets of consumption peaks, the proposed tool implements a *Formal Concept Analysis* (FCA) algorithm [9, 10]. Thus, given a set of *N* consumption peaks, it is possible determine in an automated way the maximum set of common methods involved in the *N* peaks (that is to say, the common candidates to cause the analyzed peaks).

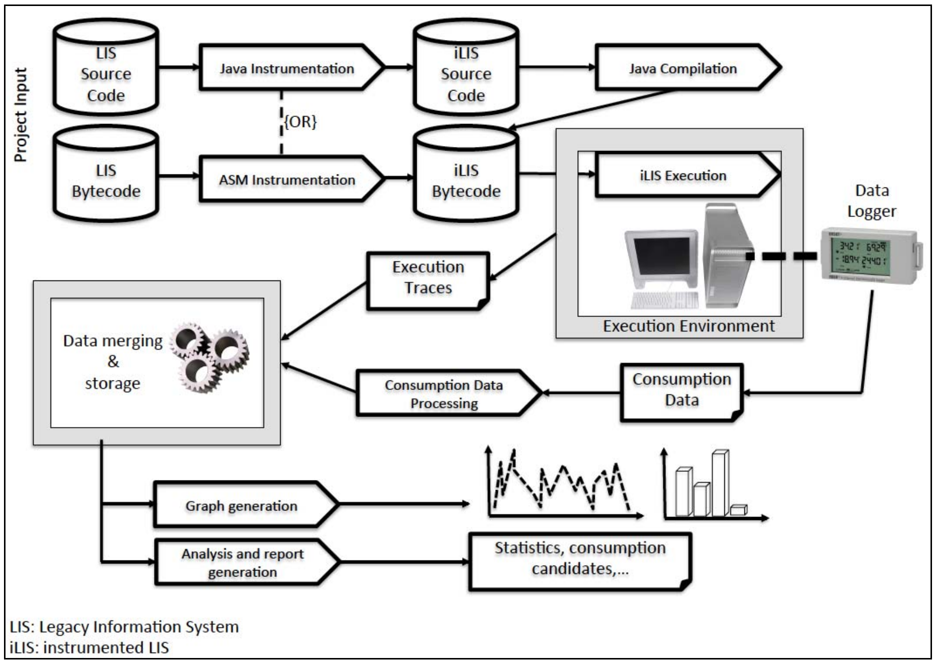


Fig. 5. Measurement process

IV. Case Study

A. *Context of the Legacy System*

The consumption measurement could be carried out using as subject of study a Java system. Nonetheless, the validation will be made taking GreeSoM as the subject of the validation (that is to say, GreeSoM will instrument, execute and analyze itself). Thus, it is possible to assess the energy consumption of our own approach.

The proposed tool is composed of 231 Java classes with a size of 4697 KB. To ease the analysis of the study, the execution will be driven by the four functionalities/scenarios drawn in section III (project creation, project instrumentation, project execution, and data analysis/chart generation).

B. *Execution of the Case Study*

After launching GreeSoM, the following activities are carried out in order to validate the approach:

1) A new project is created: This project consists on all the source code files of GreeSoM.
2) Instrumentation of the legacy system: Since GreeSoM is a Java-based system, it is possible instrument it using the built-in parser. After instrumenting GreeSoM (iGreeSoM hereinafter), the source code is compiled (compilation is carried out by a Java development environment). It is important to point out that not all the classes of GreeSoM were instrumented since some of then correspond to external libraries (such as the Java parser intended to analyze the source code in the instrumentation stage). GreeSoM allow the navigation in the created project in order to choose the files to be instrumented.
3) Execution: GreeSoM launches iGreeSoM, and then, the four scenarios mentioned are executed. For the execution of iGreeSoM, a toy Java system is used as "legacy information system". The execution of the instrumented toy Java system took approximately 150 seconds.
4) Data analysis: After executing the toy example, iGreeSoM loads the consumption and execution logs and generates the graphs corresponding to the execution of the toy Java system.

After finishing the forth step, the consumption and execution logs of the instrumented proposed tool are obtained, and the graphs corresponding graphs generated (see **Fig. 6** and **Fig. 7**).

C. *Analysis of the results*

On one hand, the consumption graph of iGreeSoM is represented in **Fig. 6**. In this graph the consumption values regarding each of the four functionalities has been highlighted by dividing the whole graph into 5 segments using 4 (red) vertical lines:

- First interval represents the consumption of the initial tasks executed in the launch process of iGreeSoM (e.g. drawing the user interface). It is possible to check that in the beginning seconds, a small peak is present (03:19:58). This peak would be due to the initial load of the application from "hard-drive"-to-"memory".
- Second interval represents the consumption due to the iGreeSoM project creation functionality. In this interval, the consumption of iGreeSoM fluctuates due to the copying process executed to replicate the legacy system (the Java toy example) in order to avoid any modification in the original legacy system. In the creation stage, the Java toy example is also instrumented. The most noteworthy consumption peak takes place in the time 03:20:14, and probably is due to the instrumentation tasks.
- Third interval represents Java toy example execution stage. This is the most time consuming stage, since the engineer is playing with the functionalities of the legacy system (Java toy example). According to the meaning of the colours of the consumption graph (see **Fig. 6**), it can be concluded that while the Java toy example was running, iGreeSoM has barely been executing any instruction (which is in turn the expected behavior). This interval finishes when the Java toy example execution finishes and the execution log file with all the execution traces is closed.
- Forth interval represents represent the location and processing of consumption and execution of log files. The two peaks probably represent the consumption of the instructions to process the records of the log files in order to save them into the database to facilitate their processing.
- Fifth interval represents the graphs generation in iGreeSoM. This task is probably the most energy-consuming task, since all the information of the log files must be processed and merged in order to build the graphs. This fact could be observed very high peak as well as in the fluctuation of the consumption in the interval.

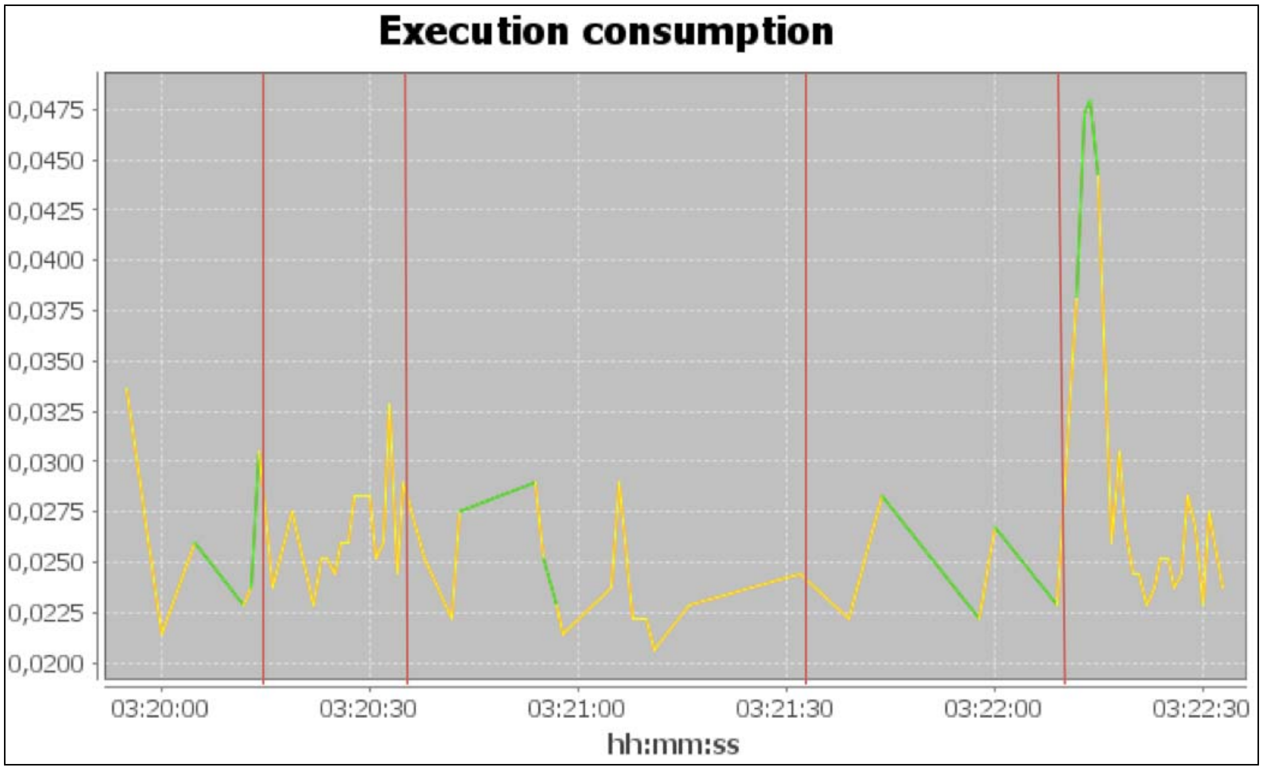


Fig. 6. Case study consumption graph

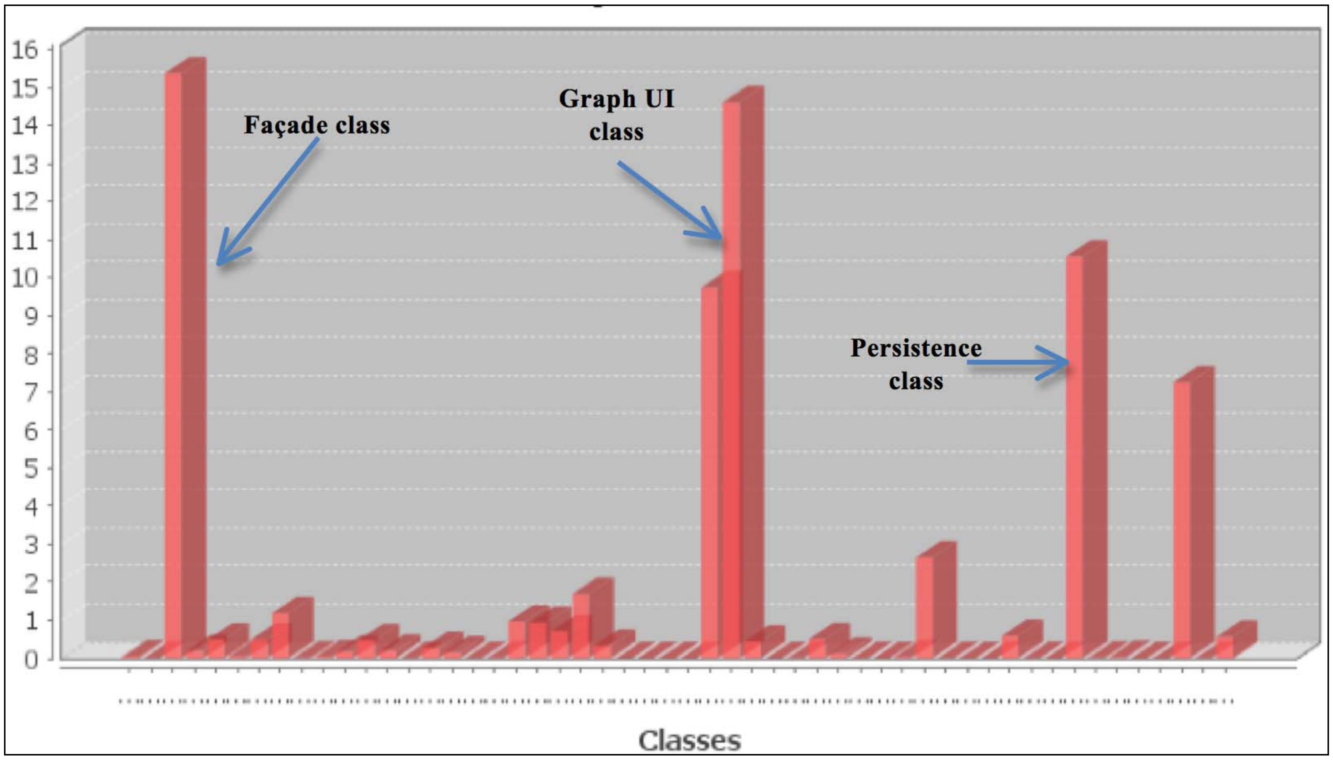


Fig. 7. Execution Time per class distribution graph

Regarding the execution graph (see **Fig. 7**), which present the total time that each class has been running (0 to 16 seconds) it is possible to draw the following conclusions regarding the execution time:

- The most time-consuming class is the Façade class. This makes sense since the Façade class is running during all the time that the tool is being used.
- The second time-consuming class is the class in charge of the graph management, since this class binds all the tasks regarding the graph management together. Also resizing or making zoom functionalities are accessed from the graph UI, contributing in this way to increase the time consumption.
- The third time-consuming class is the one implementing the persistence responsibilities. For the current case study, it makes sense, since the proposed tool is continuously saving and querying information from the database (for example in project creation, project instrumentation, log processing, graph generation, etc.).

D. *Conclusions of the case study*

Regarding this particular case study, where the subject of study is the GreeSoM system (iGreeSoM), it is observed that one of the functionalities that cause more consumption peaks is the one regarded to graph generation (execution and consumption graphs). After examining the methods related with such functionality (registered in the seconds where the consumption peak takes place) it is possible to find that the algorithm to merge information and draw the results is quite complex, with many loops, and database accesses.

This observation together with the fact that this class is also a time consuming class, presents such class as a perfect candidate to be refactored in order to improve the GreeSoM sustainability.

V. CONCLUSIONS

In this paper, authors propose a software tool intended to estimate the measurement of software energy consumption in terms of energy. Despite of the fact that exist many proposals for energy measurement [4], most of them are focused on IT infrastructures (servers, network, hardware components, etc.). The presented approach implements the technique of profiling or instrumentation of source code in order to keep track of the execution of a given legacy system. The instrumented legacy system produces log files with detailed information about its execution. While the system is executed, the energy consumption of the system where legacy software is running is measured using a hardware data logger. This hardware measures the consumption of the whole system and produces log files with the energy consumption (measured in amperes). The execution information is merged with the consumption information in order to obtain energy consumption graphs where the energy peaks can be observed. Despite the current measurement unit is the ampere, there exist other energy measurement units such as joule of kWh that will be addressed in the future.

In spite of the energy measurement is based on the whole system, it is possible to query the methods of the legacy involved in the period of time where a energy peak has occurred.

The objective of the presented approach is not the precise energy measurement of a legacy system, but the analysis of the energy consumption evolution while the legacy system is running. The analysis of this information give us the chance of determine which parts of the legacy system are suspicious of an excessive energy consumption, and in turn, offers the chance to take any kind of corrective action.

In addition, the proposed approach is intended to serve as an analysis tool in order to detect structures and patterns in the software source code that are responsible of energy consumption peaks (e.g. excessive database access, hard-drive access of intensive user-interface interaction). While other proposals are focused on detecting real power consumption of a software system, the presented one is to detect energy consumption peaks in order to find software structures that would cause such energy peak.

Finally, future lines in the presented approach are to improve the quality of the analysis of the execution and consumption information in order to provide with more accurate information. The execution of additional and more complex case studies would lead us to develop a catalog of "highly consuming practices and patterns in software source code" or in other words, "bad smells for energy consumption" in source code.

Another important research line that must be addressed is the idle-time of the instrumented legacy system. As we aforesaid, one second is a very long period of time (in terms of the number of cycles spend in compute an instruction). This fact implies that in this period not only many methods of the same legacy system could be executed, but also other processes running in the system processor can be producing power consumption peaks. At the moment, GreeSoM is able to determine (using the execution log files) when the legacy system is running and when not.

In order to improve the accuracy of the analysis carried out by GreeSoM, there exist certain energy consuming tasks that have not been taken into account, such as the energy wasted on writing the log files. This information should be recorded in order obtain more realistic values in the measurements and the obtained results. Another factor that might influence the obtained information is the Java Virtual Machine (JVM) where the legacy systems are executed. Since up to now we have not conducted any research about the energy consumption of the JVM it could be relevant to take it into account.

GreeSoM can deal only with Java systems, since code instrumentation is a very highly platform-dependent state. Nonetheless we are concerned with the fact that there exist many legacy systems developed under other platforms. So, future versions of GreeSoM should address the possibility to instrument systems developed with other languages.

Acknowledgment

This work has been funded by the MOTERO project (Consejería de Educación, Ciencia y Cultura de la Junta de Comunidades de Castilla La Mancha, y Fondo Europeo de Desarrollo Regional FEDER, PEII11- 0399-9449)

References

1. Murugesan, S., *Harnessing Green IT: Principles and Practices.* IT Professional, 2008. **10**(1): p. 24-33.
2. Murugesan, S., et al., *Fostering Green IT - Guest Editors' Introduction.* IT Professional, 2013. **15**(1): p. 16-18.
3. Naumann, S., et al., *The GREENSOFT Model: A reference model for green and sustainable software and its engineering.* Sustainable Computing: Informatics and Systems, 2011. **1**(4): p. 294-304.
4. Noureddine, A., R. Rouvoy, and L. Seinturier, *A Review of Energy Measurement Approaches.* ACM SIGOPS Operating Systems Review journal, 2013. **47**(3): p. 42-49.
5. Noureddine, A., R. Rouvoy, and L. Seinturier, *A Review of Energy Measurement Approaches.* Operating Systems Review, 2013. **47**(3): p. 42 - 49.
6. Amsel, N. and B. Tomlinson. *Green tracker: a tool for estimating the energy consumption of software*. in *Proceeding of the CHI '10 Extended Abstracts on Human Factors in Computing Systems*. 2010. Atlanta, Georgia, USA: ACM New York.
7. Kuleshov, E. *Introduction to the ASM 2.0 Bytecode Framework*. 2005; Available from: http://asm.ow2.org/doc/tutorial-asm-2.0.html [Last access: 10/03/2015].
8. Kruchten, P., et al., *Technical debt: towards a crisper definition report on the 4th international workshop on managing technical debt.* Software Engineering Notes, 2013. **38**(5): p. 51-54.
9. Wolff, K.E. *A First Course in Formal Concept Analysis*. in *Proceedings of the SoftStat'93*. 1993.
10. Tonella, P. and M. Ceccato. *Aspect Mining through the formal concept analysis of execution traces*. in *In Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04)*. 2004. IEEE Computer.
11. Johann, T., Dick, M., Naumann, S. and E. Kern, *How to measure energy-efficiency of software: Metrics and measurement results*, in Proceedings of the First International Workshop on Green and Sustainable Software (GREENS). 2005, IEEE Computer Society. pp. 51 - 54