The Development of Data Collectors in Open-Source System for Energy Efficiency Assessment

Daniel Atonge(✉), Vladimir Ivanov, Artem Kruglov, Ilya Khomyakov, Andrey Sadovykh, Dragos Strugar, Giancarlo Succi, Xavier Zelada Vasquez, and Evgeny Zouev

Innopolis University, Innopolis, Russia
d.atonge@innopolis.university

Abstract. The paper is devoted to the development of the data collectors for Windows OS and MacOS. The purpose of these plugins is to collect the process metrics from the user's device and send it to the back-end for further processing. The overall open source framework is aimed at energy efficiency analysis of the developing software products. The development presented here as a sequence of the life cycle stages, including requirements analysis, design, implementation and testing. Specifics of the implementation for each targeted operating system are given.

Keywords: Process metrics · Energy metrics · Collector · Windows · MacOS · Open source software

1 Introduction

Modelling the energy consumption of applications, gathering valid data from active and passive application processes (i.e., applications in focus and idle applications) is a crucial activity which can be used to find correlations and trends in various areas of research such as developer's productivity, applications with the highest energy consumption profiles and more. To this aim, researchers have proposed hardware-based tools as well as model-based [13] and software-based [14] techniques to approximate the actual energy profile of applications. However, all these solutions present their own advantages and disadvantages. Hardware-based tools are highly precise, but at the same time their use is bound to the acquisition of costly hardware components. Model-based tools require the calibration of parameters needed to correctly create a model on a specific hardware device. Software-based approaches are cheaper and easier to use than hardware-based tools, but they are believed to be less precise. We present the collectors (Windows, MacOs), a software-based approach whose duty is to gather all valuable

Supported by the Russian Science Foundation grant No 19-19-00623.

Published by Springer Nature Switzerland AG 2020
V. Ivanov et al. (Eds.): OSS 2020, IFIP AICT 582, pp. 14–24, 2020.
https://doi.org/10.1007/978-3-030-47240-5_2

data about active and passively running applications together with energy based metrics. This data when collected can later be processed on the server and result in precise trends and predictions which could enormously benefit software development teams. Making it Open Source exhibits several advantages, as evidenced in several scientific venues [7,9,15,21].

2 Collector Development

The development life-cycle of the collector applications were broken down into the various parts.

2.1 Analysis

Before analysis of the application and design methods, we had a series of requirements both functional and non-functional that lead to the implementation of the current application [6,22,23,26]. These requirements established the services that the client requires from our system and the constraints under which our system operates and is developed.

The functional requirements are the following:

- The list of collected metrics should be easily modifiable
- The metrics should be collected and send to DB automatically after authorization
- The time interval to send the data to server should be specifiable
- The collector should support automatic updates
- Clients should be able to send error reports
- Energy related metrics should be collected
- Product metrics should be collected
- Collectors should implement search functionality

Similarly, the non-functional requirements for the collector are the following:

- Modifiability
- Maintainability
- Adaptability
- Security
- Reusability
- Reliability

With these requirements in place, analysis of the developed application and documentation of the design decisions were made.

Analysis is that iterative process that continues until a preferred and acceptable solution or product emerges. During the analysis of our system, factual data was collected (i.e. what is lacking, what was done, what is needed, etc.). Understanding the processes involved, identifying problems and recommending feasible suggestions for improving the systems' functioning was done. This involved

studying logical processes, gathering operational data, understanding the information flow, finding out bottlenecks and evolving solutions for overcoming the weaknesses of the system so as to achieve the organizational goals for the collector. Furthermore, subdividing of complex processes involving the entire system and the identification of data store and manual processes where made (Fig. 1).

During the early stages of development, we felt the need of a simple but purposeful representation of our entire system. This representation was needed in order to:

- Specify the context of our system
- Capture system requirements
- Validate systems architecture
- Drive implementation and generate test cases

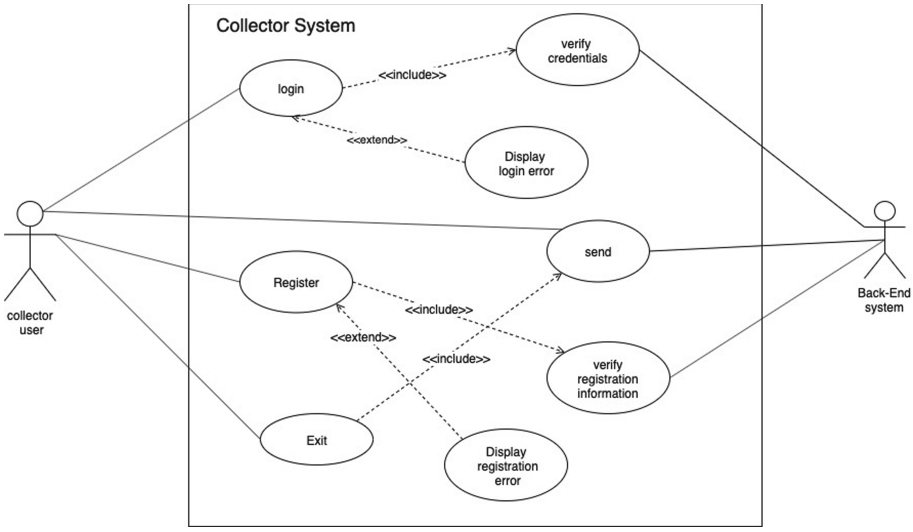

Fig. 1. Use case diagram for the collector

This was more of a thinking process and involved the creative skills. We attempted to give ideas to an efficient system that satisfies the current needs of our clients and has scope for future growth within the organizational constraints.

Overall we followed an agile development process [5,12,18,24,25], also employing techniques for Internet-based working [16], and organizing our development using a component-based approach [25].

2.2 Design

The system was designed to satisfy the requirements stated in the specifying stage. The requirements identified during Requirements Analysis were then

transformed into a System Design Document [10] that accurately describes the design of the system and that can be used as an input to system development (see Fig. 2).

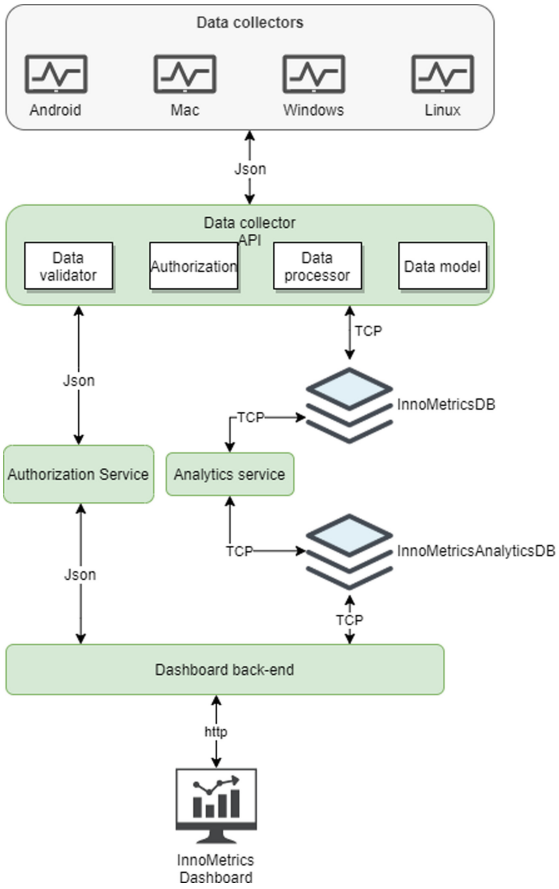

Fig. 2. Logical system design

2.3 Implementation

The logical design was then implemented to build a workable system. This demands the coding of design into computer understandable language, i.e., programming language (in our case native and specific to the targeted platform). Here, program specifications were converted into computer instructions (programs). The programs written coordinate the data movements and control the entire process in our system. Having maintainability as one of our non-functional requirements, code was well written to reduces the testing and maintenance effort. This helps in fast development, maintenance and future changes,

if required. We used Visual Studio Code 2019 and the C# programming language (for Windows OS), Xcode and Swift 4.0 (for MacOs) and lastly, QtCreator and C++ (for Linux).

We aim at gathering valuable data related to each running application process. We choose the following; process name, process id, status (app focus or idle), start time, end time, ip address, mac address, process description, processor, hard disk, memory, network and input/output usage.

The internal implementation includes external packages installed via NuGet like RestSharp, A powerful API that assist our application with sending request and reading responses from the server (in windows).

The collector is presently an application with the following interfaces:

- Registration interface for new users
- The Login interface for existing users and
- The Collector Interface: Which displays data collected from the host's machine

2.4 Testing

Before deciding to install our system and put it into operation, a test run of the system was done removing all the bugs as necessary. The output of all our test matched expected results. We ran the two categories of test:

- Program test: Referring to our requirements for expected results of our system, coding, compiling and running our executable was the routine. After our locally based collector application was up and running, each use-case of our system was tested to match expected outcome. We carried out various verification and noted unforeseen happenings which were eventually corrected.
- System test: After carrying out program test and errors removed, running the overall system with the back-end and making sure it meets the specified requirements was done. Our system was fully developed and ready for usage by clients.

3 Integration with the Back-End System

In order to be able to perform authenticated requests, such as sending the collected metrics, collectors should first authenticate with the back-end. That is performed using the POST HTTP request to the back-end system. Then, back-end API requests the Auth Token from the Authentication Service, which is also a part of the back-end system. Then, the Auth Service gets the user information from the database, and if everything turns out to be alright, responds with the token to the collectors.

Then, collectors perform the needed actions described in the sections above, i.e. primarily store the activities along with energy efficiency metrics locally. When the time comes to send the report to the back-end, the collector sends a

POST HTTP request with the Auth Token to the back-end containing all the information about processes that were in use.

Back-end services, upon receiving the request from the collectors, validate the authenticity of the user and receives, and stores in the database the data sent by the collector.

The sequence diagram (see Fig. 3) showcases the above mentioned scenario, adding to the picture the Innometrics Dashboard, a go-to tool for managers to have an overview of the entire reporting process [11].

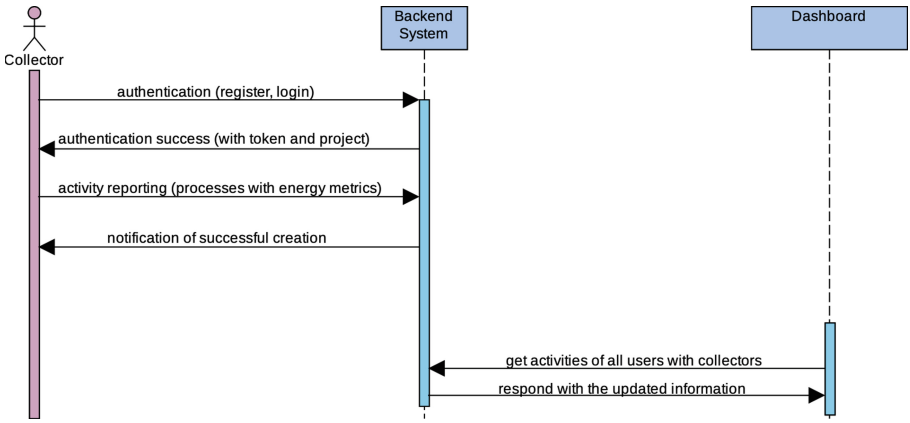

Fig. 3. Logical system design

4 Implementation for the Different OS

This section provides with the peculiarities of developing the collectors for different types of OS. For now the collectors for Windows OS and MacOS are developed. Further, the collectors for X11 and Android OS are planned to be developed.

4.1 Specific of Windows OS Collector

The collector is compose of 3 forms which will come up occasionally (i.e. Account verification (at login) form, Registration form, Collector form). These are the only means of communication with the system from a user perspective.

In Code, best programming practices where used such as those specified by the object-oriented, service-oriented paradigm. Furthermore, good code related practices where enforced such as the DRY (Do Not Repeat Yourself) principle, Naming conventions and more [8].

The main interface (Fig. 4), consists of table view, which is automatically updated by a running service to make sure our application does not block at any time and that, the user can always interact with the collector.

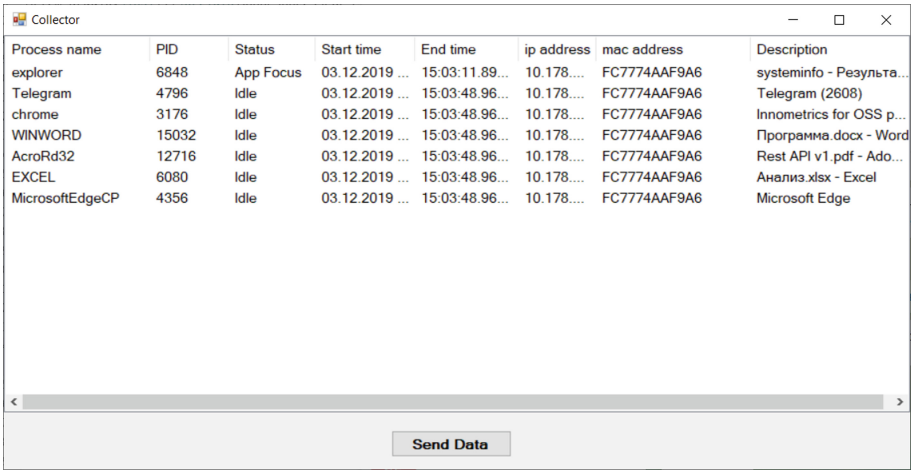

Fig. 4. Logical system design

Data being collected by running services are the following; process name, process id, status (app focus or idle), start time, end time, ip address, mac address, process description, battery consumption. All other details have been well presented in the sections above.

Battery draining applications result in bad user experience and dissatisfied users. Optimal battery usage (energy usage) is an important aspect that every client must consider.

Application energy consumption is dependent on a wide variety of system resources and conditions. Energy consumption depends on, but is not limited to, the processor the device uses, memory architecture, the storage technologies used, the display technology used, the size of the display, the network interface that you are connected to, active sensors, and various conditions like the signal strength used for data transfer, user settings like screen brightness levels, and many more user and system settings.

For precise energy consumption measurements one needs specialized hardware. While they provide the best method to accurately measure energy consumption on a particular device, such a methodology is not scalable in practice, especially if such measurements have to be made on multiple devices. Even then, the measurements by themselves will not provide much insight into how your application contributed to the battery drain, making it hard to focus on any application optimization efforts [4].

The collector aims at enabling users to estimate their application's energy consumption without the need for specialized hardware. Such estimation is made possible using a software power model that has been trained on a reference device representative of the low powered devices applications might run on.

Using the PerformanceCounter, PerformanceCounterCategory and many more related classes (made available by the .NET Framework [3]), the energy

usage can be computed. Performance counter(s) provided information such as CPU time, Total Processor Time per process, CPU usage, Memory usage, network usage and more. The MSDN documentation [2] was used to better understand how we could utilise the available components in attaining our goal (collecting energy consumption metrics).

It is difficult to match up constantly changing application process IDs and names. Imperfection in designed power model as energy consumption depends on a variety of factors not limited to those we can collect using these available performance classes.

4.2 Specific of MacOS Collector

The developed MacOS application is a status menu bar application, as shown in Fig. 5.

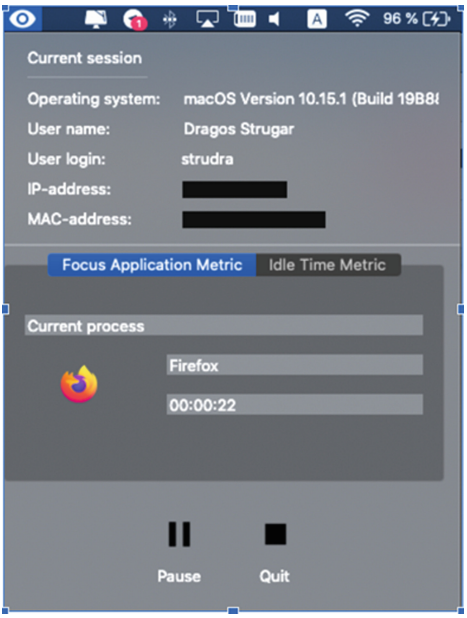

Fig. 5. Logical system design

The application shows the most important information about the user that is collecting the activities. It shows the currently running process, along with the time it is active. In background, it stores this data locally and prepares to send it to the back-end service, as described above.

A collection of information about the currently running process can be obtained using the NSProcessInfo class in the Task Management subsection of Foundation API. This includes thermal state, app performance as well as other

specifics. It allows developers to track the information about an activity in the currently running application. However, the problem here arises when we want to access such information for other processes; MacOS has security built-in that does not allow our application to monitor such information.

MetricKit [1] came as a stable solution, that allowed us to aggregate and analyze per-device reports on power and performance metrics, particularly important for prediction [17,19,20]. However, what it does it collects metrics and sends them once every 24 h, and is thus not fully suitable to our case.

Thus, we have decided to exclude the CPU utilization, while other metrics, like battery power, memory, GPU and others are already in development and testing stage.

5 Conclusion

Our profiling method and the tools we have available are only able to attribute energy consumption at process level. Any finer granularity, although desirable, is not possible.

Hardware resource usage could fill the gap when it comes to accurately relating EC to individual software elements hence enabling us to compute the UEC.

Profiling the performance requires basic understanding of the hardware components that has to be monitored through "performance counters" in windows and when interpreting performance data for further analysis, context information has to be taken into account (e.g. hardware-specific details).

To evaluate Unit Energy Consumption we can monitor the following hardware resources [14]: Hard disk: disk bytes/sec, disk read bytes/sec, disk write bytes/sec Processor: % processor usage Memory: private bytes, working set, private working set Network: bytes total/sec, bytes sent/sec, bytes received/sec IO: IO data (bytes/sec), IO read (bytes/sec), IO write (bytes/sec)

Attributing some weights to elements of the UEC or by some reliable assumption such as considering the power model to be linear in nature for each individual component, We can compute the SEC Metric.

Reporting these metrics is also useful in identifying potential trade-offs between energy efficiency and other aspects of software quality (e.g. maintainability)

Acknowledgments. The work presented in this paper has been performed thanks to the support by the Russian Science Foundation grant No 19-19-00623.

References

1. Metrickit documentation. https://developer.apple.com/documentation/metrickit. Accessed 12 Apr 2019
2. Microsoft, performance counters. https://docs.microsoft.com/en-us/windows/win32/perfctrs/performance-counters-portal. Accessed 12 May 2019

3. Performance counters in the .net framework. https://docs.microsoft.com/en-us/dotnet/framework/debug-trace-profile/performance-counters. Accessed 12 May 2019
4. Acar, H., Alptekin, G.I., Gelas, J., Ghodous, P.: The impact of source code in software on power consumption. IJEBM **14**, 42–52 (2016)
5. Coman, I.D., Robillard, P.N., Sillitti, A., Succi, G.: Cooperation, collaboration and pair-programming: Field studies on backup behavior. J. Syst. Softw. **91**, 124–134 (2014)
6. Corral, L., Sillitti, A., Succi, G.: Software assurance practices for mobile applications. Computing **97**(10), 1001–1022 (2015). https://doi.org/10.1007/s00607-014-0395-8
7. Di Bella, E., Sillitti, A., Succi, G.: A multivariate classification of open source developers. Inf. Sci. **221**, 72–83 (2013)
8. Dooley, J.: Software Development and Professional Practice. Apress, Berkeley (2011)
9. Fitzgerald, B., Kesan, J.P., Russo, B., Shaikh, M., Succi, G.: Adopting open source software: A practical guide. The MIT Press, Cambridge (2011)
10. Hao-yu, W., Hai-li, Z.: Basic design principles in software engineering. In: 2012 Fourth International Conference on Computational and Information Sciences, pp. 1251–1254 (2012)
11. Ivanov, V., Larionova, D., Strugar, D., Succi, G.: Design of a dashboard of software metrics for adaptable, energy efficient applications. J. Vis. Lang. Comput. **2019**(2), 145–153 (2019)
12. Janes, A., Succi, G.: Lean software development in action. Lean Software Development in Action, pp. 249–354. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-00503-9_11
13. Kalaitzoglou, G., Bruntink, M., Visser, J.: A practical model for evaluating the energy efficiency of software applications. In: ICT4S (2014)
14. Kor, A.L., Pattinson, C., Imam, I., AlSaleemi, I., Omotosho, O.: Applications, energy consumption, and measurement. In: 2015 International Conference on Information and Digital Technologies. IEEE (2015)
15. Kovács, G.L., Drozdik, S., Zuliani, P., Succi, G.: Open source software for the public administration. In: Proceedings of the 6th International Workshop on Computer Science and Information Technologies (2004)
16. Maurer, F., Succi, G., Holz, H., Kötting, B., Goldmann, S., Dellen, B.: Software process support over the Internet. In: Proceedings of the 21st International Conference on Software Engineering. ICSE 1999, pp. 642–645. ACM (1999)
17. Musílek, P., Pedrycz, W., Sun, N., Succi, G.: On the sensitivity of COCOMO II software cost estimation model. In: Proceedings of the 8th International Symposium on Software Metrics. METRICS 2002, pp. 13–20. IEEE Computer Society (2002)
18. Pedrycz, W., Russo, B., Succi, G.: A model of job satisfaction for collaborative development processes. J. Syst. Softw. **84**(5), 739–752 (2011)
19. Pedrycz, W., Russo, B., Succi, G.: Knowledge transfer in system modeling and its realization through an optimal allocation of information granularity. Appl. Soft Comput. **12**(8), 1985–1995 (2012)
20. Ronchetti, M., Succi, G., Pedrycz, W., Russo, B.: Early estimation of software size in object-oriented environments a case study in a CMM level 3 software firm. Inf. Sci. **176**(5), 475–489 (2006)

21. Rossi, B., Russo, B., Succi, G.: Modelling failures occurrences of open source software with reliability growth. In: Ågerfalk, P., Boldyreff, C., González-Barahona, J.M., Madey, G.R., Noll, J. (eds.) OSS 2010. IAICT, vol. 319, pp. 268–280. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13244-5_21
22. Scotto, M., Sillitti, A., Succi, G., Vernazza, T.: A relational approach to software metrics. In: Proceedings of the 2004 ACM Symposium on Applied Computing. SAC 2004, pp. 1536–1540. ACM (2004)
23. Sillitti, A., Janes, A., Succi, G., Vernazza, T.: Measures for mobile users: an architecture. J. Syst. Architect. **50**(7), 393–405 (2004)
24. Sillitti, A., Succi, G., Vlasenko, J.: Understanding the impact of pair programming on developers attention: a case study on a large industrial experimentation. In: Proceedings of the 34th International Conference on Software Engineering. ICSE 2012, pp. 1094–1101. IEEE Press (2012)
25. Sillitti, A., Vernazza, T., Succi, G.: Service oriented programming: a new paradigm of software reuse. In: Gacek, C. (ed.) ICSR 2002. LNCS, vol. 2319, pp. 269–280. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-46020-9_19
26. Vernazza, T., Granatella, G., Succi, G., Benedicenti, L., Mintchev, M.: Defining metrics for software components. In: Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics. vol. XI, pp. 16–23 (2000)