Combining profiling and monitoring to analyze test coverage and identify performance problems

Nico Beierle
Berner&Mattner Systemtechnik GmbH
Berlin, Germany
nico.beierle@berner-mattner.com

Peter M. Kruse
Berner&Mattner Systemtechnik GmbH
Berlin, Germany
peter.kruse@berner-mattner.com

ABSTRACT

The use of profilers is a common approach for locating bottlenecks in software performance. Existing profilers typically generalize memory consumption and CPU usage. This work is dedicated to profiling-based identification of performance problems for specific moments of program execution. By combining conventional profiling with monitoring of user actions (e.g. mouse and keyboard inputs), a more fine-grained analysis of program behavior is possible. The calculation of coverage levels for GUI tests will also be available. The current state of this work describes a proposed solution. Realization of a prototype implementing the approach is currently ongoing.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

Keywords

Performance Measures, Monitoring

1. INTRODUCTION

When improving the performance of a program, it is necessary to initially locate its bottlenecks. The use of profilers is a common practice and a helpful approach. These mostly extensive analysis tools monitor various system properties, such as memory or CPU usage, as well as information from the current program, such as method calls. In the subsequent evaluation phase, the previously collected measured values are combined to give the user a statistical picture. Part of this overview are statistics about data types and method calls, but also diagrams illustrating the use of resources over time. However, this deliberately generalized illustration also leads to a loss of information, since the state information of the program at a particular time can not be considered further. Nevertheless, this information would be helpful. Thus, in some cases it would be good to know, for example, which method calls or which user actions at a given time have led to the increase in resource consumption. In particular, with regard to automatic test execution, when the program execution is not observed by the user, this information would be useful in order to easily identify error sources.

This work should therefore be dedicated to the question of how to identify, based on profiling mechanisms, performance problems as precisely as possible in a specific scenario.

2. GOALS

The aim of this work is to implement a profiling tool for recording the actual program flow, especially the interplay of actions and reactions in order to later be able to identify parts of the program with potential performance issues as precisely as possible. Actions considered here are all events that provoke the invocation of methods, so as user input, but also the method calls itself. Reactions typically would be the called methods. The use of resources (memory usage and processor load) is of particular interest. Figure 1 illustrates the results of recording in a graphical representation. The graph shows the CPU usage as a function of time. The markings *Button1* and *Button2* are intended to easily identify user input at a certain time. This approach helps to easily identify performance issues from the user's view. It also allows to analyze the program flow from top-down without initially taking the code base into consideration, which might lead to confusion because of too many additional information.

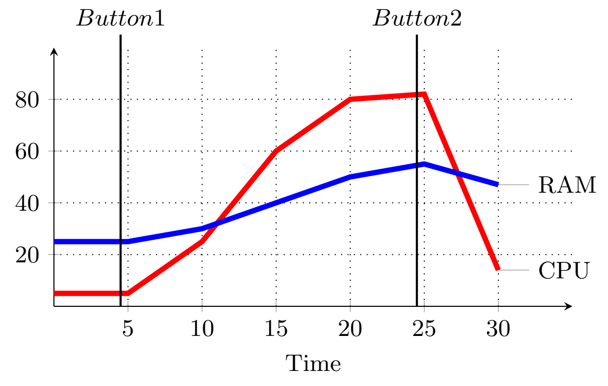

Figure 1: Resources

VALUETOOLS 2015, December 14-16, Berlin, Germany
Copyright © 2016 ICST
DOI 10.4108/eai.14-12-2015.2262590

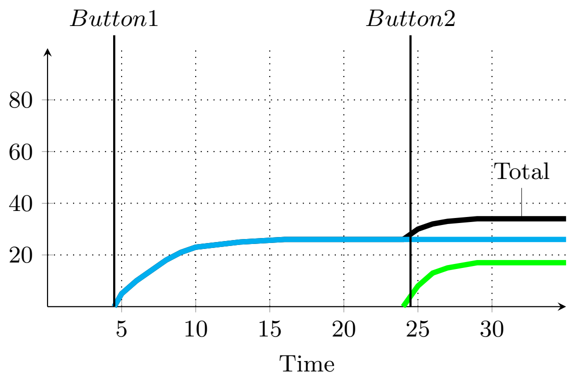

Figure 2: Code Coverage

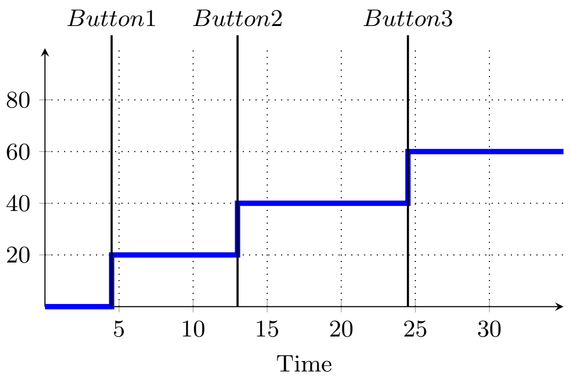

Figure 3: GUI Coverage

Furthermore, the combined reporting of user activities and method calls allows a detailed monitoring of the code coverage during test execution. Figure 2 shows the result of such coverage analysis. The coverage rate for method calls can be determined separately for each user action. The total coverage (e.g. of elements in the graphical user interface (GUI)) can also be reported (Figure 3).

Looking at test environments, this would allow to assess the effectiveness of each test step within a test case, or the effectiveness of the test suite as a whole.

3. WORK BREAKDOWN

To complete the above-mentioned objectives, several logging mechanisms has to be implemented. These include the tracking of system characteristics such as memory usage and CPU utilization, the program flow (method calls) and the logging of user interactions. In the second step, the data needs further processing and synchronization. Part of this step is the measurement of the GUI coverage as well as the code coverage. Finally a user interface must be implemented for the graphic representation of the results. Future tasks can involve further data analyzes like the comparison of different measurements or the identification of memory leaks.

4. RELATED WORK

For conventional profiling, there is broad body of tools available [3]. The combination with user interaction at the GUI is out of focus. There is some existing work on combining profiling and monitoring [5]. Jovic et al. describe an approach for the automation of performance testing of interactive java applications [4].

While conventional capture and replay (C&R) testing also focuses on monitoring user actions [7, 2], the intention is different. In C&R the user actions are recorded for later replay only. There is no analysis on coverage levels of GUI elements. Since C&R targets GUI testing only, the underlying code remains hidden from the testing process. While this is in accordance to black-box testing procedures, it might not be very beneficial for performance testing, esp. for identifying and analyzing bottlenecks.

While there are approaches for monitoring GUI coverage [8] and also structured test design for GUI testing [6], the performance dimension has not yet been fully explored. For approaches targeting the GUI performance [1], coverage is out of focus.

5. CONCLUSIONS

In this work, a combination of conventional profiling with monitoring of user actions (e.g. mouse and keyboard inputs) is proposed. This allows for a more fine-grained analysis of program behavior and also for a calculation of coverage levels (e.g. in GUI tests). The approach is currently a proposal, realization of a prototype implementing the approach is currently ongoing.

6. REFERENCES

[1] A. Adamoli, D. Zaparanuks, M. Jovic, and M. Hauswirth. Automated gui performance testing. *Software Quality Journal*, 19(4):801–839, 2011.

[2] S. Arlt, C. Bertolini, S. Pahl, and M. Schäf. Trends in model-based gui testing. *Advances in Computers*, 86:183–222, 2012.

[3] J. Du, N. Sehrawat, and W. Zwaenepoel. Performance profiling of virtual machines. *ACM SIGPLAN Notices*, 46(7):3–14, 2011.

[4] M. Jovic, A. Adamoli, D. Zaparanuks, and M. Hauswirth. Automating performance testing of interactive java applications. In *Proceedings of the 5th Workshop on Automation of Software Test*, pages 8–15. ACM, 2010.

[5] M. Jovic and M. Hauswirth. Measuring the performance of interactive applications with listener latency profiling. In *Proceedings of the 6th international symposium on Principles and practice of programming in Java*, pages 137–146. ACM, 2008.

[6] P. M. Kruse and O. Stadie. Closing Gaps between Capture and Replay: Model-based GUI Testing. In *1st International Workshop on User Interface Test Automation*, 2015.

[7] A. M. Memon. Gui testing: Pitfalls and process. *Computer*, 35(8):87–88, 2002.

[8] T. E. Vos, P. M. Kruse, N. Condori-Fernández, S. Bauersfeld, and J. Wegener. Testar: Tool support for test automation at the user interface level. *International Journal of Information System Modeling and Design (IJISMD)*, 6(3):46–83, 2015.