Software Impacts 11 (2022) 100220

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

# ProcessPerformance: A portable and easy-to-use tool to measure resource consumption of running processes

Miguel Garcia [a], Jose Quiroga [a], Francisco Ortin [a,b,*]

[a] University of Oviedo, Computer Science Department, Federico Garcia Lorca 18, 33007, Oviedo, Spain
[b] Munster Technological University, Department of Computer Science, Rossa Avenue, Bishopstown, Cork, Ireland

ARTICLE INFO

*Keywords:*
Runtime performance
CPU
Memory
Network traffic
Resource consumption

ABSTRACT

The measurement of the resources consumed by an application at runtime is an important task in different scenarios such as program optimization, malware and bug detection, and hardware scaling. Although different tools exist for this purpose, they sometimes show some limitations such as operating system and hardware dependencies, performance overhead, and usage complexity. For this reason, we create ProcessPerformance, a portable and easy-to-use command-line tool that provides information about the CPU, memory, and network resources consumed by any combination of running processes. It also avoids the performance overhead caused by software and binary code injection.

## Code metadata

| | |
|---|---|
| Current code version | 1.1.1 |
| Permanent link to code/repository used for this code version | https://github.com/SoftwareImpacts/SIMPAC-2021-199 |
| Permanent link to Reproducible Capsule | https://codeocean.com/capsule/7889504/tree/v1 |
| Legal Code License | MIT |
| Code versioning system used | git |
| Software code languages, tools, and services used | C# 8.0 |
| Compilation requirements, operating environments & dependencies | .Net Core 3.1+, TraceEvent 2.0.55 |
| If available Link to developer documentation/manual | https://github.com/ComputationalReflection/ProcessPerformance/blob/master/README.md |
| Support email for questions | garciarmiguel@uniovi.es |

## 1. Introduction

It is sometimes necessary to measure the resources a process is consuming at runtime [1]. That measurement is a valuable piece of information to optimize applications consuming too many resources, scale the hardware resources depending on the running processes, identify potential malicious programs, and compare resource consumption of different processes [1]. Common resources the user needs to measure are CPU, memory, and network consumption [2]. Resource consumption data are usually collected by hardware monitors, additional routines implemented at the operating system level, or code injected in a program [3]. The choice of a measurement technique depends on different factors, such as the data to be measured, the potential impact of the measurement tools on the performance of the whole application, and the available hardware and software resources, among others [3].

Modern operating systems include system monitors to supervise the usage of system resources in a computer [4]. Task managers are system monitor programs that provide information about computer performance, including the name of running processes, CPU and GPU load, I/O information, logged-in users, and operating system services [5]. Example task managers are SysInternals Process Explorer [6], GNOME System Monitor [7], and macOS Activity Monitor [8]. These task managers provide a graphical user interface to give information to the user, but they do not facilitate the extraction of the data measured. On the contrary, tasklist [9], top [10], iotop [11] and nethogs [12] are textual command-line task managers that make it easier to retrieve that

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author at: University of Oviedo, Computer Science Department, Federico Garcia Lorca 18, 33007, Oviedo, Spain.
*E-mail addresses:* garciarmiguel@uniovi.es (M. Garcia), quirogajose@uniovi.es (J. Quiroga), ortin@uniovi.es (F. Ortin).

https://doi.org/10.1016/j.simpa.2022.100220
Received 28 December 2021; Received in revised form 4 January 2022; Accepted 4 January 2022

information. iotop and nethogs only provide network information, and none of those tools support Windows, Linux, and macOS.

Software instrumentation tools add pieces of code around source or binary code to measure dynamic resource consumption [13]. They can be used to diagnose memory errors, evaluate runtime performance, generate trace information, and profile applications [1]. Valgrind is a binary instrumentation framework for different Unix-based executable files [1]. Apache Netbeans Profiler is a similar tool for Java programs [14]. The main drawback of software instrumentation tools is the memory and CPU consumption overhead they introduce with the instrumented code.

Hardware Performance Counters (HPC) is another approach to obtain detailed information about application execution [15]. HPC is based on hardware counters that register microprocessor activities upon the trace generation phase. Later, that information can be analyzed by tools such as Windows Reliability and Performance Monitor (perfmon) [16], SysInternals Process Monitor [17], and perf [18]. The main benefit of HPCs compared to software-based approaches is that HPCs provide lower performance overhead to obtain detailed performance information, but they are hardware-dependent [19].

Some other approaches are based on modifying the behavior of the virtual machine that is used to run the software. The Java Management Extensions (JMX) framework provides a configurable mechanism for managing and monitoring Java applications [20]. With Managed Beams (MBeans), programs can be instrumented to measure the runtime resources consumed by the application. Likewise, dotnet-trace is a .Net application that enables the collection of application traces without a native profiler [21]. These kinds of tools provide runtime information about runtime resource consumption, but only for applications executed on particular virtual machines.

Process and System Utilities (psutil) is a cross-platform library for retrieving information of running processes, such as CPU, memory, disks, and network consumption [22]. It is written in Python 3.4 and aims to monitor and profile systems. It supports most operating systems. psutils is delivered as an API, so users must write their own programs to visualize the resource consumption of running programs.

In this paper, we present ProcessPerformance, an open-source multiplatform tool to monitor and retrieve the CPU, memory, and network resources consumed by any combination of running processes. Runtime process instrumentation is performed at the operating-system level, so no overhead is produced by high-level source code instrumentation. HPC information may be used by the operating software when the hardware supports HPC, causing almost no runtime overhead [19]. ProcessPerformance is implemented in the free open-source .Net Core platform that runs on Windows, Linux, and macOS. It provides an easy-to-use command-line interface that does not require the subsequent analysis of tracing log information. The services of ProcessPerformance can be used by any other application, since its source code is available for download at https://github.com/ComputationalReflection/ProcessPerformance.

## 2. Application description and functions

ProcessPerformance is designed to easily provide information about the resources consumed by any processes. It retrieves information about the CPU, memory, and network resource consumption during a period of time. ProcessPerformance is a portable open-source application implemented for the .Net Core platform. Memory and CPU usage is measured with the `Process` class in the `System.Diagnostics` namespace, which supports interaction with local and remote processes, event logs, and performance counters [23]. For process network consumption, we use the `TraceEvent` library [24] that allows us to collect and process event data of the processes running on the operating system. To get the information of the overall network traffic, ProcessPerformance uses the `NetworkInterface` class that provides configuration and statistical information for a network interface.

What follows is a brief description of how ProcessPerformance gathers resource consumption information from the operating system, as illustrated in Fig. 1:

- **CPU consumption**, measured as the percentage of use of the total CPU resources. It is computed with the following equation:

$$CPU\ consumption = total\ processor\ time\ /\ clock\ time\ /\ number\ of\ system\ cores \quad (1)$$

  The first operand is the `TotalProcessorTime` property of the `Process` class, which returns the time that the microprocessor spends working on a task (it is the sum of user and privileged processor time) [23]. Processor times are represented as blue rectangles in Fig. 1. Clock time is the elapsed execution time for the time interval measured (i.e., $start\ time - end\ time$ in Fig. 1). The division of these two values gives us the percentage of CPU used by the process. That value is then divided by the number of system cores (the `ProcessorCount` property of the `Environment` class) to obtain the CPU consumption, because `TotalProcessorTime` represents the sum of working times for all the cores.
- **Overall network traffic** refers to the number of bytes transferred across the network by all the processes since ProcessPerformance is executed. The `GetIPv4Statistics` method is used to get that information from the operating system, including the bytes sent (`BytesSent` property of `IPv4InterfaceStatistics`, shown as dark green rectangles in Fig. 1) and received (`BytesReceived` property, shown as light green rectangles in Fig. 1). With this information, ProcessPerformance displays both the number of bytes transferred and the transmission rate.
- **Process network consumption** is the number of bytes transferred by one single process across the network since ProcessPerformance started measuring. A `TraceEventSession` object is used to register all the TCP/IP events triggered in the system. It follows the Observer design pattern, where listeners are registered to be notified when different events occur [25]. ProcessPerformance registers itself for the `TcpIpRecv` and `TcpIpSend` events to store the information about data received and sent. Each time one of these two events is triggered, we check whether the process causing the data transfer is the one to be monitored and, if so, we update the variables counting the number of bytes transmitted by that process. Fig. 1 shows the data sent by a process with dark gray rectangles, while light gray boxes represent the data received.
- **Memory consumption**. To measure the memory consumed by a process at runtime, we count the maximum size of working set memory used by a process (i.e., the `PeakWorkingSet` property of `Process` [26]). The working set of a process is the set of memory pages currently visible to the process in physical RAM memory. Those pages are resident and available for an application to be used without triggering a page fault. The working set includes both shared and private data. The shared data comprises the pages that contain all the instructions that the process executes, including those from the process modules and the system libraries. As shown in Fig. 1, it is common that the working set memory used by a process grows at runtime as the program demands more memory from the operating system.

### 2.1. Usage and examples

ProcessPerformance is an easy-to-use command-line tool. If the user runs[1] `ProcessPerformance -help` the following command-line options are described:

[1] In Windows, the application is directly run with `ProcessPerformance`. In Unix-based operating systems, it must be written `dotnet ProcessPerformance.dll`.

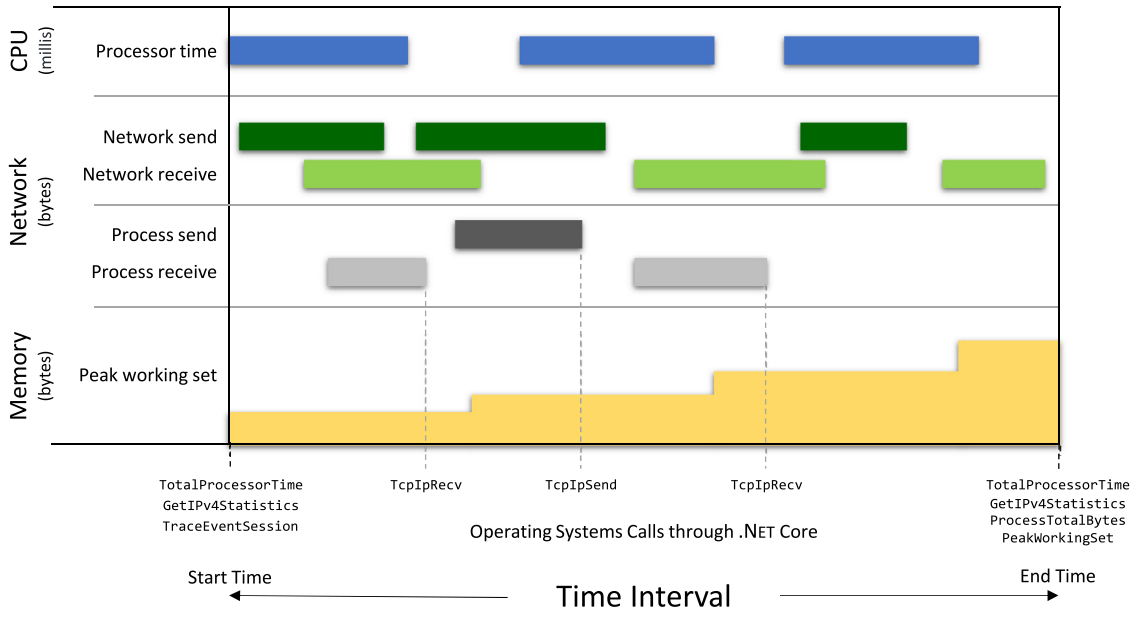


**Fig. 1.** Gathering resource consumption information from the operating system.

```
chrome (4 processes):
      CPU: 5.18%
    | Memory: 3.425 MB
    | Process: Sent 29 KB (116 kbps) - Received 29 KB (116 kbps)
chrome (4 processes):
      CPU: 2.12%
    | Memory: 3.761 MB
    | Process: Sent 46 KB (68 kbps) - Received 96 KB (268 kbps)
chrome (4 processes):
      CPU: 3.38%
    | Memory: 3.999 MB
    | Process: Sent 93 KB (188 kbps) - Received 201 KB (420 kbps)
...
```

**Fig. 2.** Output for `ProcessPerformance chrome -interval:500`.

- $process_1$ $process_2$ ... $process_n$. A space-separated list of the names or PIDs (process identifiers) of the processes to be monitored. If no process is passed, the overall system resources are displayed.
- `-interval:`*milliseconds*. The interval used to gather the runtime information of resource consumption, expressed in milliseconds. The default value is 1,000 (one second).
- `-network:`*IP_address*. IP address of the network interface used to measure data transmission.
- `-csv`. Shows the output in comma-separated values (CSV) format.
- `-help`. Describes the different command-line arguments.

For example, the following command shows the CPU, memory, and network resources consumed by the Chrome web browser, every half second: `ProcessPerformance chrome -interval:500`. That command produces the output displayed in Fig. 2, where ProcessPerformance tells us that there are four different processes running Chrome, and displays the resources consumed by the four processes.

ProcessPerformance allows measuring the resources consumed by a complex application or system, defined as a collection of running programs. Therefore, the following command can be used to measure overall resources used by a client–server application that runs an application server (Apache Tomcat), two persistence systems (PostgreSQL and Neo4j), and the Chrome web browser as a client:

`ProcessPerformance chrome tomcat neo4j postgres -network:192.168.137.2`

The output is shown in Fig. 3. The network traffic is displayed with two values: the sum of all the data transferred by the 22 processes (4 programs), and the data transmitted through the 192.168.137.2 network interface.

## 3. Impact

When measuring the resources consumed by an application at runtime, it is important to define a statistically rigorous methodology and the correct use of tools [27]. The characteristics of ProcessPerformance have made it a valuable tool to measure the runtime resources consumed by many kinds of applications. In the scenario of comparing techniques for the implementation of programming languages, ProcessPerformance has been used to measure runtime execution and memory consumption of program specialization [28], static single assignment (SSA) transformations [29], hybrid dynamic and static typing [30], compiler implementation [31], runtime type cache optimization [32], intersection and union types [33], and type inference [34]. It has also been used to compare the efficiency of different Python implementations [35], the `invokedynamic` opcode included in Java 7 [36], the implementation of dynamic languages for the Java platform [37], and the adaptability of Java applications [38].

```
chrome + tomcat + neo4j + postgres (22 processes):
      CPU: 7.71%
    | Memory: 3,064 MB
    | Processes: Sent 39 KB (312 kbps) - Received 45 KB (360 kbps)
    | Network: Sent 93 KB (744 kbps) - Received 373 KB (2,984 kbps)
chrome + tomcat + neo4j + postgres (22 processes):
      CPU: 8.15%
    | Memory: 3,332 MB
    | Processes: Sent 40 KB (8 kbps) - Received 46 KB (8 kbps)
    | Network: Sent 154 KB (488 kbps) - Received 688 KB (2,520 kbps)
chrome + tomcat + neo4j + postgres (22 processes):
      CPU: 2.18%
    | Memory: 3,334 MB
    | Processes: Sent 41 KB (8 kbps) - Received 46 KB (0 kbps)
    | Network: Sent 178 KB (192 kbps) - Received 695 KB (56 kbps)
...
```

**Fig. 3.** Output for `ProcessPerformance chrome tomcat neo4j postgres -interval:500 -network:192.168.137.2`.

In the scenario of aspect-oriented programming (AOP) ProcessPerformance has been utilized to compare the efficiency of dynamic and static weavers for the Java [39] and .Net platforms [40], to analyze the suitability of AOP for distributed systems security [41], and to measure the runtime performance of the DSAW AOP platform [42].

Likewise, our tool has measured memory and CPU consumption of various virtual machine implementations, such as the addition of structural intercession [43] and dynamic inheritance to .Net [44], and the implementation of the nitrO virtual machine [45]. ProcessPerformance has been used to measure network and memory consumption in the design and implementation of mobile applications for multiple platforms [46], including the DIMAG back-end module [47] and the LIZARD native interface generator [48].

We have used ProcessPerformance to measure the network traffic generated by an infrastructure to deliver synchronous programming laboratories online [49]. In this case, various applications were measured together, since the system comprises different processes. ProcessPerformance was a helpful tool to know the network traffic generated by the entire system, which was a critical aspect due to the limited Internet connections the students have in their households [49].

ProcessPerformance has also been utilized to measure training and inference time of machine learning models in different scenarios such as programmer classification [50], students' performance prediction [51], decompilation [52,53], and analysis of binary files [54]. We have used our tool to compare the runtime resources consumed by different persistence systems such as graph databases [55], reflective persistence systems [56], aspect-oriented database evolution systems [57], and orthogonal object-based persistence [58].

## 4. Limitations

As mentioned in Section 2, ProcessPerformance uses the `TraceEvent` library to know the data sent and received by a particular process. `TraceEvent` was originally designed to parse the Event Tracing for Windows (ETW) events generated by the Windows operating system. Thus, ProcessPerformance only provides this particular information when run on a Windows operating systems. However, it is worth noting that the traffic information of the overall network is provided for any operating system.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

This work has been partially funded by the Spanish Department of Science, Innovation and Universities: project RTI2018-099235-B-I00. The authors have also received funds from the University of Oviedo, Spain through its support to official research groups (GR-2011-0040).

## References

[1] N. Nethercote, J. Seward, Valgrind: A program supervision framework, Electron. Notes Theor. Comput. Sci. 89 (2) (2003) 44–66.
[2] V. Salapura, K. Ganesan, A. Gara, M. Gschwind, J.C. Sexton, R.E. Walkup, Next-generation performance counters: Towards monitoring over thousand concurrent events, in: ISPASS 2008 - IEEE International Symposium On Performance Analysis Of Systems And Software, 2008, pp. 139–146.
[3] J.R. Larus, T. Ball, Rewriting executable files to measure program behavior, Softw. Pract. Exp. 24 (2) (1994) 197–218.
[4] C.A.R. Hoare, Monitors: An operating system structuring concept, Commun. ACM 17 (10) (1974) 549–557.
[5] K. Pothuganti, A. Haile, S. Pothuganti, A comparative study of real time operating systems for embedded systems, Int. J. Innov. Res. Comput. Commun. Eng. 6 (4) (2016) 12008–12014.
[6] Microsoft, Process explorer v16.43, 2021, https://docs.microsoft.com/en-us/sysinternals/downloads/process-explorer.
[7] The GNOME Project, System monitor, 2021, https://help.gnome.org/users/gnome-system-monitor/stable.
[8] Apple Inc., Activity monitor user guide for mac OS Monterey, 2021, https://support.apple.com/guide/activity-monitor/welcome/mac.
[9] Microsoft, Tasklist, 2021, https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/tasklist.
[10] tOp. linux manual page, 2021, https://man7.org/linux/man-pages/man1/top.1.html.
[11] G. Chazarain, iotop. linux manual page, 2021, https://www.man7.org/linux/man-pages/man8/iotop.8.html.
[12] A. Engelen, Nethogs, 2021, https://github.com/raboof/nethogs.
[13] J. Pierce, M.D. Smith, T. Mudge, Instrumentation tools, in: Fast Simulation Of Computer Architectures, Springer, US, Boston, MA, 1995, pp. 47–86.
[14] I. Kostaras, C. Drabo, J. Juneau, S. Reimers, M. Schröder, G. Wielenga, Debugging and profiling, in: Pro Apache NetBeans, Springer, 2020, pp. 127–178.
[15] L. Uhsadel, A. Georges, I. Verbauwhede, Exploiting hardware performance counters, in: 5th Workshop On Fault Diagnosis And Tolerance In Cryptography, 2008, pp. 59–67.
[16] Microsoft, Perfmon, 2021, https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/perfmon.
[17] Microsoft, Process monitor v3.86, 2021, https://docs.microsoft.com/en-us/sysinternals/downloads/procmon.
[18] Perf: Linux profiling with performance counters, 2021, https://perf.wiki.kernel.org/index.php.
[19] C. Malone, M. Zahran, R. Karri, Are hardware performance counters a cost effective way for integrity checking of programs, in: Proceedings Of The Sixth ACM Workshop On Scalable Trusted Computing, in: STC'11, Association for Computing Machinery, New York, NY, USA, 2011, pp. 71–76.

[20] H. Kreger, Java management extensions for application management, IBM Syst. J. 40 (1) (2001) 104–129.
[21] Microsoft, Dotnet-trace performance analysis utility, 2021, https://docs.microsoft.com/en-us/dotnet/core/diagnostics/dotnet-trace.
[22] Psutil. cross-platform lib for process and system monitoring in python, 2021, https://github.com/giampaolo/psutil.
[23] Microsoft, Process.TotalProcessorTime property (system.diagnostics), 2021, https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.process.totalprocessortime?view=netcore-3.1.
[24] Microsoft, The microsoft.diagnostics.tracing.TraceEvent library, 2021, https://github.com/Microsoft/perfview/blob/main/documentation/TraceEvent/TraceEventLibrary.md.
[25] E. Gamma, R. Helm, R. Johnson, J. Vlissides, D. Patterns, Elements of Reusable Object-Oriented Software, Addison-Wesley Reading, Massachusetts, 1995.
[26] Microsoft, Process.PeakWorkingSet64 property (system.diagnostics), 2021, https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.process.peakworkingset64?view=netcore-3.1.
[27] A. Georges, D. Buytaert, L. Eeckhout, Statistically rigorous java performance evaluation, in: Proceedings Of The 22nd Annual ACM SIGPLAN Conference On Object-Oriented Programming Systems And Applications, in: OOPSLA, ACM, New York, NY, USA, 2007, pp. 57–76.
[28] F. Ortin, M. Garcia, S. McSweeney, Rule-based program specialization to optimize gradually typed code, Knowl.-Based Syst. 179 (2019) 145–173.
[29] J. Quiroga, F. Ortin, SSA transformations to facilitate type inference in dynamically typed code, Comput. J. 60 (9) (2017) 1300–1315.
[30] F. Ortin, D. Zapico, J.B.G. Pérez-Schofield, M. Garcia, Including both static and dynamic typing in the same programming language, IET Softw. 4 (4) (2010) 268–282.
[31] M. Garcia, F. Ortin, J. Quiroga, Design and implementation of an efficient hybrid dynamic and static typing language, Softw. Pract. Exp. 46 (2) (2016) 199–226.
[32] J. Quiroga, F. Ortin, D. Llewellyn-Jones, M. Garcia, Optimizing runtime performance of hybrid dynamically and statically typed languages for the .NET platform, J. Syst. Softw. 113 (2016) 114–129.
[33] F. Ortin, M. García, Union and intersection types to support both dynamic and static typing, Inform. Process. Lett. 111 (6) (2011) 278–286.
[34] F. Ortin, Type inference to optimize a hybrid statically and dynamically typed language, Comput. J. 54 (11) (2011) 1901–1924.
[35] J.M. Redondo, F. Ortin, A comprehensive evaluation of common python implementations, IEEE Softw. 32 (4) (2014) 76–84.
[36] P. Conde, F. Ortin, JINDY: a java library to support invokedynamic, Comput. Sci. Inf. Syst. 11 (1) (2014) 47–68.
[37] F. Ortin, P. Conde, D. Fernandez-Lanvin, R. Izquierdo, The runtime performance of invokedynamic: An evaluation with a java library, IEEE Softw. 31 (4) (2013) 82–90.
[38] I. Lagartos, J.M. Redondo, F. Ortin, Efficient runtime metaprogramming services for java, J. Syst. Softw. 153 (2019) 220–237.
[39] O. Rodriguez-Prieto, F. Ortin, D. O'Shea, Efficient runtime aspect weaving for java applications, Inf. Softw. Technol. 100 (2018) 73–86.
[40] J.M. Felix, F. Ortin, Efficient aspect weaver for the .NET platform, IEEE Lat. Am. Trans. 13 (5) (2015) 1534–1541.
[41] M. Garcia, D. Llewellyn-Jones, F. Ortin, M. Merabti, Applying dynamic separation of aspects to distributed systems security: a case study, IET Softw. 6 (3) (2012) 231–248.
[42] F. Ortin, L. Vinuesa, J.M. Felix, The DSAW aspect-oriented software development platform, Int. J. Softw. Eng. Knowl. Eng. 21 (7) (2011) 891–929.
[43] F. Ortin, M.A. Labrador, J.M. Redondo, A hybrid class- and prototype-based object model to support language-neutral structural intercession, Inf. Softw. Technol. 44 (1) (2014) 199–219.
[44] J.M. Redondo, F. Ortin, Efficient support of dynamic inheritance for class- and prototype-based languages, J. Syst. Softw. 86 (2) (2013) 278–301.
[45] F. Ortin, D. Diez, Designing an adaptable heterogeneous abstract machine by means of reflection, Inf. Softw. Technol. 47 (2) (2005).
[46] P. Miravet, I. Marín, F. Ortin, J. Rodríguez, Framework for the declarative implementation of native mobile applications, IET Softw. 8 (2014) 19–32.
[47] P. Miravet, I. Marín, F. Ortin, A. Rionda, DIMAG: A framework for automatic generation of mobile applications for multiple platforms, in: Proceedings Of The 6th International Conference On Mobile Technology, Application And Systems, in: Mobility'09, 2009, pp. 1–8.
[48] I. Marin, F. Ortin, G. Pedrosa, J. Rodriguez, Generating native user interfaces for multiple devices by means of model transformation, Front. Inf. Technol. Electr. Eng. 16 (12) (2015).
[49] M. Garcia, J. Quiroga, F. Ortin, An infrastructure to deliver synchronous remote programming labs, IEEE Trans. Learn. Technol. 14 (2) (2021) 161–172.
[50] F. Ortin, O. Rodriguez-Prieto, N. Pascual, M. Garcia, Heterogeneous tree structure classification to label java programmers according to their expertise level, Future Gener. Comput. Syst. 105 (2020) 380–394.
[51] M. Riestra-González, M. del Puerto Paule-Ruí z, F. Ortin, Massive LMS log data analysis for the early prediction of course-agnostic student performance, Comput. Educ. 163 (2021) 104108–104128.
[52] J. Escalada, T. Scully, F. Ortin, Improving type information inferred by decompilers with supervised machine learning, 2021, arXiv:2101.08116.
[53] F. Ortin, J. Escalada, Cnerator: A python application for the controlled stochastic generation of standard c source code, SoftwareX 15 (2021) 100711–100717.
[54] J. Escalada, F. Ortin, T. Scully, An efficient platform for the automatic extraction of patterns in native code, Sci. Program. 2017 (2017).
[55] O. Rodriguez-Prieto, A. Mycroft, F. Ortin, An efficient and scalable platform for java source code analysis using overlaid graph representations, IEEE Access 8 (2020) 72239–72260.
[56] F. Ortin, B. Lopez, J.B. García Perez-Schofield, Separating adaptable persistence attributes through computational reflection, IEEE Softw. 21 (6) (2004).
[57] R.H. Pereira, J.B.G. Perez-Schofield, F. Ortin, Modularizing application and database evolution – an aspect-oriented framework for orthogonal persistence, Softw. Pract. Exp. 47 (2) (2017) 193–221.
[58] J. García Perez-Schofield, E. García Roselló, F. Ortin, M. Pérez Cota, Visual zero: A persistent and interactive object-oriented programming environment, J. Vis. Lang. Comput. 19 (3) (2008).