DE GRUYTER OPEN

Applied Computer Systems

doi: 10.1515/acss-2014-0015

2014/16

Specification of the Framework for Evaluating the Efficiency of Mechanisms for Concurrent Processes

Vladislav Nazaruk[1], Pavel Rusakov[2], [1–2]*Riga Technical University, Latvia*

***Abstract* – The goal of this paper is to give a specification of the software framework that evaluates the efficiency of different mechanisms for concurrent processes, notably process synchronization mechanisms. The paper discusses the concept of this framework, the potential users of it and some necessary considerations, including assumptions. Further, it defines general requirements for the framework and presents its desired conceptual design. The conclusions and possible directions for future work end the paper.**

***Keywords* – Concurrent computing, efficiency, inter-process communication, mechanisms, software framework.**

I. Introduction

Nowadays, a vast number of operating systems (including mobile and embedded operating systems such as those based on the Linux kernel, Apple iOS, Microsoft Windows Phone and Windows Embedded) and software platforms and/or frameworks (for example, Microsoft .NET Framework, Java Platform) provide a native capability of concurrent computing, also known as multi-tasking. The main principle of concurrent computing is that several tasks (processes, threads) are executed either simultaneously or quasi-simultaneously (alternately on the same processing unit).

Despite some possible costs of applying and using the concurrency in a specific computer program, it can facilitate a better software architecture (including more understandable logics – especially when the original problem that the program solves is concurrent by its nature) and, mostly notable, allow the executing environment to execute other tasks on the same processing unit while one (or some) of the tasks is idle (for example, is waiting for some external event). The latter advantage means that the environment is able to reallocate processor time between tasks dynamically when any thread becomes idle. This mostly decreases the amount of time (not the total processor time!) needed for the tasks to be executed – compared to executing the tasks sequentially (i.e., non-concurrently). Thus, one of the main effects of using the concurrent computing is the increase of the performance (and, therefore, the efficiency) of software systems [1].

In concurrent systems, if there are several processes (threads) that are to achieve a common goal – we will call them a process group – then these processes should communicate with each other (directly or indirectly) in a certain way. (In other words, assuming a process is a vertex of a graph, and a communication between two processes is a directed edge (showing the initiator and the target of the communication) between corresponding vertices, we can define a process group as an arbitrary connected directed graph.) Such a communication between processes is realized by the means of the use of one or several inter-process communication (IPC) mechanisms – external objects that provide the interfaces to be called by the processes. Here we describe the model of the inter-process communication where IPC mechanisms are passive, and only communicating processes are active. (Other cases can be narrowed down to this model by correspondingly defining the scope of each IPC mechanism.)

Despite all IPC mechanisms are designed to achieve a common general goal of inter-process communication, there are different types of these mechanisms, for example, process synchronization mechanisms, message-passing mechanisms, shared memory. Each type of these mechanisms has different instances, which can further vary by their implementation specifics. For example, there are different classes of blocking process synchronization mechanisms (such as locks, monitors, condition variables, barriers, rendezvous – with different possibilities of their implementation); and at the same time there are a number of non-blocking process synchronization mechanisms [2]. This shows that many of IPC mechanisms are to a certain degree interchangeable.

Therefore, the same concurrent computing problem can be implemented in a number of ways, i.e., by using different IPC mechanisms. Given the same process group, we can tell that the actual IPC mechanisms used for the communication have an effect on the overall performance of the process group. Such an impact of the IPC mechanisms used on the overall performance is the *object* of the current research. This dependency, when concerning the increase of the overall performance of the concurrent system, is rather topical, especially in high-performance computing and other applications where a large number of computations shall be performed.

In [3], we showed that for measuring the impact of process synchronization mechanisms on the overall performance of the system, a special software framework can be used. In that paper, we also defined the general concept of such a framework. In the current paper, we continue to develop and refine the concept of the framework. Thus, the main *goal* of this paper is to give a specification of the framework for evaluating the efficiency of IPC mechanisms. This includes specifying general requirements for the framework and the conceptual design of it. Here, we also extend the scope of the

framework that was initially defined – from process synchronization mechanisms to IPC mechanisms.

By specifying the framework, we are trying to free people who are concerned about the performance of the concurrent systems from the necessity to perform a detailed analysis of the possible impacts of IPC mechanisms on the performance of these systems.

Section I of the current paper introduces the research and shows the motivation behind it and its background. Section II describes the related studies and shows the distinction between them and the current research. Section III provides the specification of the framework for evaluating the efficiency of IPC mechanisms, giving the view on different aspect of the framework. Section IV concludes the research and states the directions for the possible future work.

II. Related Work

The necessity to verify or estimate the efficiency and other non-functional characteristics of the newly created inter-process communication mechanisms is obvious, when they are suggested but not compared with alternatives like in [4], or the validation is possible only using case studies like in [5]. However, the difficulty of modeling and thus simulating inter-process communication is mentioned in [6], which methodology is planned to be implemented in the prototype of the framework for performance prediction and power estimation of computer based systems.

The necessity of evaluation of efficiency of communication mechanisms as part of the entire performance measurement is recognized in high performance computing (HPC) systems [7]–[10]. HPC systems built on multicore processors should provide excellent performance on shared memory, distributed memory and hybrids to parallel systems [8]. For example, the high-performance communication mechanism is vital in virtual machine (VM) networks [7], where VMs may be distributed or allocated to the same computer while performing one transactional task. As the authors mention in [9], it is greatly desired that performance of parallel applications would be predicted already at the design stage. However, like the authors in [6] and [10], they also note difficulties in application modeling and simulation, especially when resource contention can significantly affect the application performance.

Performance analysis is also very important for mobile device applications. For example, the authors in [11] use improved performance assertions for verification of the performance requirements (lower levels, middleware and applications) of complex multitask software systems that mobile devices may contain. Furthermore, the authors in [12] indicate the importance of performance estimation and efficient concurrency for the power consumption purpose.

The above-mentioned related studies show the need for evaluating the efficiency of IPC mechanisms. They also show that such an evaluation is useful on different hardware platforms, including high-performance computing and mobile platforms.

The authors in [13] suggest the analysis of causal dataflows for concurrent programs that use multi-core technology. The suggested framework includes a developed formal control-flow model for the programs in the form of control nets (a kind of Petri nets), a definition of dataflow analyses based on causal flows in a program, and algorithms for the analysis that explore the partially-ordered runs of the program (thus avoiding creation of the global state space for a program), as well as efficient algorithms for the class of distributive causal concurrent dataflow problems (such as reaching definitions, coverability checking and available expressions analysis). The usage of Petri nets allows using Petri net analysis tools for the analysis of concurrent program control nets. However, the defined framework is not validated for large programs. The authors have used only small experimental programs for a preliminary analysis of the framework.

Extended Queuing Network Models (EQNs) are widely used in analytical performance modelling [6]; however, they are able to model a behavior of some application with the very important restriction. The restriction is that distinct jobs of an EQN cannot communicate with each other, and hence modelling of parallel applications with inter-process communication is not possible. The authors in [6] extend EQNs with two novel network elements, namely, trigger nodes and delay nodes, and measure non-functional characteristics in that extended network model.

The performance analysis of high-performance communication mechanisms is also validated in the experimental way executing communication scenarios on predefined hardware platforms [7] or by using "a set of micro-benchmarks and application level benchmarks on an IBM BG/P, Cray XT, and InfiniBand cluster" like in [8].

In case of mobile applications, both programmers and testers need a mechanism to analyze performance requirements that may be distributed at different levels of applications, middleware and devices themselves. The authors in [11] suggest using assertion programming languages and corresponding testing tools, which verify assertions specified in code by triggering them online. As the authors mention, a group of other approaches uses a post-mortem principle, where a complete (offline) or partial (semi-online) log file is analyzed after the application is terminated. This is not suitable for complex mobile systems.

The authors in [9] propose a framework named PHANTOM for measuring the performance of large-scale parallel applications. They note that execution time of such applications depends on several factors, which include sequential computation time in each process, communication time and their convolution. In the framework, they implement two main techniques, namely, deterministic sequential replay and concurrent replay of representative processes of the application on a single node at real speed and a trace-driven network simulation. The concurrent replication of the parallel application is also mentioned in [14].

The authors in [15] describe a framework that is based on a modular performance analysis (MPA), which is able to compute hard upper and lower bounds during performance

estimation for various performance criteria in a real-time system. The MPA differs from both probabilistic estimation techniques and simulations.

In this paper, we give a specification of the framework that tries to incorporate some characteristic ideas of the solutions described in the related studies. However, we try to follow the principle that we defined in [3] – "the full (…) framework should be able to handle concurrent systems specified on different abstraction levels – for each abstraction level of the model of the concurrent system being analyzed. This implies that the framework should be organized in several layers." Therefore, we do not limit our framework with a specific implementation; moreover, we want the framework to be customizable to the extent possible. However, is should not be too much abstract, and it should have at least one working implementation.

III. Specification of the Framework

As we stated it in Section I, the actual mechanisms used for the inter-process communication have an effect on the overall performance of the corresponding process group (and the entire system as well). This correlation by itself can raise different questions, and ideally, the framework should deal with all of those. These questions include:

- How to evaluate the impact of a specific IPC mechanism on the entire system performance?
- When does a specific IPC mechanism show better/worse results?
- How to select the best fitting mechanism for a particular case?

A. User Classes and Their Goals

The questions mentioned above let define the following two potential user classes for the framework:

- a developer of an IPC mechanism,
- a potential user of an IPC mechanism.

The first class of users typically is a developer of a new IPC mechanism (or a person who improves an existing IPC mechanism). He can have several goals of using the framework, including the following ones:

- to estimate, how efficient the IPC mechanism developed is in different cases; this is usually to be done by comparing the IPC mechanism developed with other [existing] IPC mechanisms;
- to estimate, how the efficiency of the IPC mechanism developed depends on the specific case;
- to estimate, which cases are the most appropriate for using the IPC mechanism developed.

The second class of users typically is a potential user of an existing IPC mechanism, for example, a programmer or a software architect who needs to select a specific IPC mechanism for the use in the system being developed. His main goal of using the framework is to have assistance in choosing the most efficient (or at least rather efficient) IPC mechanism for the use in the specific system.

B. Aspects of Efficiency (Main Outputs of the Framework)

In this paper, we use the term *efficiency of the IPC mechanism* (in a particular case) as a measure that corresponds to the overall performance of the particular concurrent system, when the processes in this system use the specified IPC mechanism for communicating with each other. This is considered to be the main output of applying the framework.

When concerning the performance of the concurrent program, some authors (for example, in [16]) analyze mostly the speed of execution of the program. However, there is a number of different metrics that can be used as an objective function for the evaluation of the performance of the system. Some of them are specific to concurrent computing, and some of them are common for both sequential and concurrent computing. We define the following typically minimum amount of metrics that can (and even should) be used when determining the objective function for the performance of a concurrent program:

- overall execution time of the program – a time interval between the start and the end of execution of the program; this metric typically is important on desktop and other non-mobile systems, where power consumption makes no or almost no difference,
- active execution time (CPU time) of the program – the sum of processor time intervals, where the corresponding processing units were busy with executing the program (its processes or the IPC mechanism); this metric typically is important on mobile and embedded systems, where power consumption is of importance, and it is proportional to CPU-busy time,
- probability of deadlocks – a metric that shows how frequently deadlocks will occur – when blocking IPC mechanisms (such as locks, see [2]) are used.

Some other performance metrics we have described in [3].

Due to the nature of concurrent systems, the performance metrics cannot be evaluated precisely – in a way that the value obtained will be nearly the same when measurements are repeated. Therefore, for each metric (including the objective function) we propose the need to obtain the following values:

- expected (mean) value,
- variance,
- possible intervals of the value.

Exactly the collection of such values gives a better evaluation of performance of a concurrent system and, therefore, of efficiency of an IPC mechanism.

When analyzing the efficiency, it is needed to be stated that it weakly correlates with the complexity of each method of a mechanism. It rather strongly correlates with external events (from the viewpoint of an IPC mechanism), including the pattern of calling the methods, and *some* of their parameters.

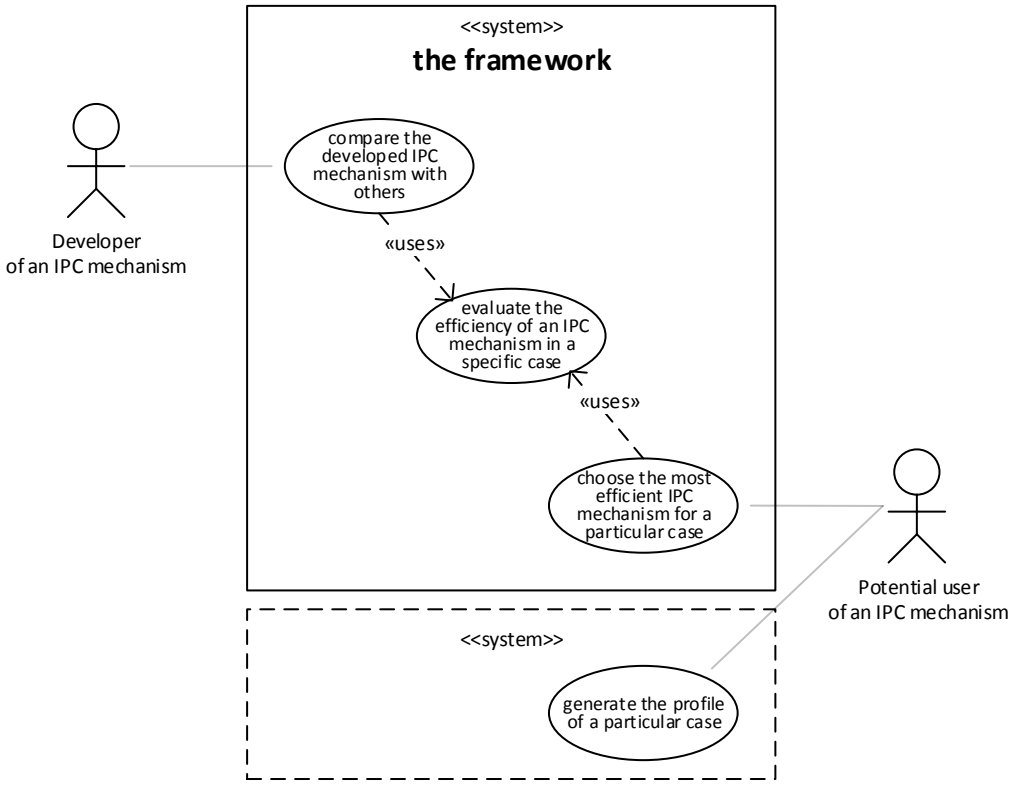

Fig. 1. The high-level use-case model of the proposed framework for evaluating the efficiency of IPC mechanisms. The model is given in the UML Use Case diagram notation.

C. Parameters that Influence the Output

The questions mentioned at the beginning of Section III support the fact we stated in [3] – that given a specific process group, the evaluation (or measurement) of the efficiency of the IPC mechanism used can differ in each execution case. The possible difference in the evaluation results can be induced by the following factors [3]:

- there are changes in input data;
- the concurrent system (consisting of the process group) depends not only on its input data, but also on its persistent internal state, which has been changed;
- the concurrent system is executed in a different environment;
- the concurrent system or its execution environment is non-deterministic.

One of the main cases of application of the efficiency evaluations obtained by the use of the framework is to see, which IPC mechanism and concurrent system in general (combining with its environment) can afford better performance. Moreover, the results should give quantitative information about the degree of difference between the measurements [3].

Therefore, the framework should be reliable enough in order to make acceptable conclusions despite possible influencing factors mentioned above [3]. In order to reach that, these factors are described here.

The first factor (changes in input data) is a natural factor for a possible change in the result of the performance measurement. Moreover, this factor can affect measurement results significantly. In order to minimize the possible effect of this factor, the comparison of measurement results should only be done for systems with the same input data. Therefore, the framework should be used to analyze concurrent systems of the same behavior, and only on the same input data set. Therefore (considering the framework is denoted by *F*), the framework should take an input data *I* as a parameter, as well as the concurrent system *CS* itself:

$$F(CS;I):\ \langle CS;I\rangle \rightarrow M, \tag{1}$$

where *M* is a result of measurement [3].

The second possible influencing factor is the presence of an internal state of a system, which is stored in the system between its executions, and which can also affect the output. According to [3], we will consider the value of this state at the moment when the system is started as part of input data. Then, when using the framework, it is needed to compare concurrent systems not only with equal inputs, but also with equal initial states.

The third influencing factor shows the necessity for the evaluation to be performed separately for different environments (or cases).

The fourth influencing factor, that the system or its execution environment can be stochastic, is one of the main issues needed to be resolved. This issue is addressed in Section III.E (direction 3).

D. Use Cases of the Framework

Having defined the potential users of the framework with their goals and expectations, as well as requirements for the objective function of the performance, specific requirements for the framework could be concluded. We define these requirements in a form of use cases. They are the following (see Fig. 1):

- "Evaluate the efficiency of an IPC mechanism in a specific case" – a core function of the framework which results in the estimation of the efficiency of the IPC mechanism (see Section III.B);
- "Compare the developed IPC mechanism with others" – a function that uses the core function multiple times, for different IPC mechanisms (possibly, it uses the database of the evaluation results obtained previously);
- "Choose the most efficient IPC mechanism for a particular case" – a function that, given a performance profile of a specific case, chooses the best-fitting IPC mechanism; it calls the core function similarly.

The function that should be executed, but which now is supposed to be outside the framework, is to "generate the [performance] profile of a particular case". The generated profile could be then used by the corresponding functions of the framework.

In order to the core function to be able to execute rather efficiently, we define the following assumptions that we are to verify during the approbation of the framework:

- Despite the whole system is complex, we assume that for our purpose the concurrent system can be divided into several independent IPC-centered parts;
- We also assume that we do not need to analyze the logics of all processes that communicate through specific IPC mechanisms.

Having these assumptions defined, we define the necessary and presumptively appropriate inputs for the core function of the framework ("Evaluate the efficiency of an IPC mechanism in a specific case"):

- a formal description of an IPC mechanism;
- a profile/characteristics of a specific execution environment;
- probabilistic characteristics of "client processes" of the IPC mechanism, including:
  - number of processes,
  - pattern of calling specific methods of a mechanism by each/every process;
- probabilistic characteristics of the environment (for example, the number of resources available) that are not "independent", but will have an impact both on the IPC mechanism and on the processes.

E. Conceptual Design of the Framework

The problems stated in the form of questions at the beginning of Section III can be tried to be solved by going three different directions (partly defined in [3]):

- Direction 1 (theoretical analysis):
  - Make a complete formal (mathematical) model of an IPC mechanism.
  - Mathematically conclude a formula (or formulas) that describe how the overall performance depends on at least main parameters of the concurrent system.
  - Apply the concluded formula to evaluate the overall performance.
- Direction 2 (observation of real systems):
  - Take a real computer program (or write a "synthetic" program with needed properties).
  - Add to the program measurement routines which will measure the needed time intervals and other metrics.
  - Run the program a needed number of times (possibly, also with different values of its properties) to evaluate the overall performance.
- Direction 3 (simulation):
  - Make a complete formal model of an IPC mechanism.
  - Limit the main parameters of the concurrent system, in order for them to describe statistically common cases.
  - Perform computer simulation to evaluate the overall performance.

The direction 1 has an advantage that potentially there will be considered all possible cases, and the resulting estimation of the efficiency would be maximally precise. However, this direction has significant disadvantages:

- Concluding a formula can be done only by a human, and this is very resource-intensive operation, especially when trying to conclude a formula that tries to reflect the impact of many parameters of the concurrent system;
- The analysis or application of the resulting analytical formula could be harder or similar in difficulty with the initial problem of evaluating the efficiency;
- These activities should be redone manually every time when an IPC mechanism or its implementation changes (even slightly changes – for example, during the development of a mechanism).

Direction 2 has an advantage that measurements can be done relatively easy (by inserting to a program a relatively simple measurement code). However, the disadvantages are the following [3]:

- The precision of measurements will suffer due to the measurement code will interfere with the parts of base program;
- Time needed to obtain results could be reasonably large due to real-time execution; moreover, many execution times would be necessary if there is a need to obtain the measurements for different input parameters;
- There will be rather hard to generalize the obtained results due to a large number of influencing factors.

Direction 3 needs inputs defined in a special formal way, as well as an appropriate simulating environment. Nevertheless, this direction has the following advantages:

- The results will be obtained mush faster (comparing to direction 2) due to non-real-time execution of the model [3];
- Due to the use of simulation, this approach can be rather well automated;

- It would be easier to see the correlation between input information and measures, and for the analysis of the results, mathematical methods could be used (results would be more statistic-oriented) – this is mostly due to the minimization of influencing factors [3].

Taking into account the amount of manual work needed to perform the efficiency analysis, as well as potentially large number of different IPC mechanisms and environmental parameters to analyze, we conclude that Direction 3 is the most appropriate for the framework.

IV. Conclusion and Future Work

The main conclusions of the research are the following:

- In order to evaluate the efficiency of an IPC mechanism, we need to analyze it in context (in specific cases);
- For the evaluation of the efficiency, we need to appropriately determine the objective function for the performance of a concurrent program;
- Different contexts have different efficient IPC mechanisms;
- Automated analysis by the means of simulation in many cases is more resource-efficient;
- The described framework is appropriate for such an analysis.

The main direction for the possible future work is to implement the framework in a minimal amount that is necessary for approbating it. Then the efficiency of the framework itself could be measured. In addition, it could be possible to evaluate the possibility of expanding and applying the framework not only to local systems, but also to distributed systems.

References

[1] A. B. Downey, "The Little Book of Semaphores," 2nd ed., 2008. [Online]. Available: http://greenteapress.com/semaphores/downey08semaphores.pdf.

[2] V. Nazaruk, and P. Rusakov, "Blocking and Non-Blocking Process Synchronization: Analysis of Implementation," *Applied computer systems*. Vol.47, 2011, pp. 145–150. ISSN 1407-7493.

[3] V. Nazaruk, and P. Rusakov, "Measuring the Performance of Process Synchronization with the Model-Driven Approach," in *Communications in Computer and Information Science: The 19th International Conference on Information and Software Technologies (ICIST 2013): Proceedings*, Lithuania, Kaunas, October 10–11, 2013. Berlin: Springer Berlin Heidelberg, 2013, pp. 403–414, ISBN 9783642419461, e-ISBN 9783642419478, ISSN 1865-0929, e-ISSN 1865-0937. http://dx.doi.org/10.1007/978-3-642-41947-8_34

[4] Q. Ri, B. Yanru, and R. Changming, "CommMgr: A new inter-process communication management software," in *Computer Standards & Interfaces*, Volume 28, Issue 5, June 2006, pp. 572–584. http://dx.doi.org/10.1016/j.csi.2005.02.005

[5] S.-H. Hung, C.-H. Tu, and W.-L. Yang, "A portable, efficient inter-core communication scheme for embedded multicore platforms," *Journal of Systems Architecture*, vol. 57, Issue 2, February 2011, pp. 193–205, ISSN 1383-7621. http://dx.doi.org/10.1016/j.sysarc.2010.11.003

[6] A. W. Liehr, and K. J. Buchenrieder, "Simulating inter-process communication with Extended Queuing Networks," In: *Simulation Modelling Practice and Theory*, vol. 18, Issue 8, September 2010, pp. 1162–1171. http://dx.doi.org/10.1016/j.simpat.2009.08.010

[7] Y. Bai, Y. Ma, C. Luo, D. Lv, and Y. Peng, "A high performance inter-domain communication approach for virtual machines," *Journal of Systems and Software*, vol. 86, Issue 2, February 2013, pp. 367–376. http://dx.doi.org/10.1016/j.jss.2012.08.054

[8] F. Blagojević, P. Hargrove, C. Iancu, and K. Yelick, "Hybrid PGAS runtime support for multicore nodes," In: *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model (PGAS '10)*. ACM, New York, NY, USA, 2010, Article 3, p. 10. http://dx.doi.org/10.1145/2020373.2020376

[9] J. Zhai, W. Chen, and W. Zheng, "PHANTOM: predicting performance of parallel applications on large-scale parallel machines using a single node," SIGPLAN Not. vol. 45, no. 5 (January 2010), pp. 305–314. http://dx.doi.org/10.1145/1837853.1693493

[10] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Mapping on multi/many-core systems: survey of current and emerging trends," in *Proceedings of the 50th Annual Design Automation Conference (DAC '13)*. ACM, New York, NY, USA, 2013, Article 1, p. 10.

[11] R. Lencevicius, and E. Metz, "Performance assertions for mobile devices," In: *Proceedings of the 2006 international symposium on Software testing and analysis (ISSTA '06)*. ACM, New York, NY, USA, 2006, pp. 225–232. http://dx.doi.org/10.1145/1146238.1146264

[12] M. Nikitovic, and M. Brorsson, "An adaptive chip-multiprocessor architecture for future mobile terminals," in *Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems (CASES '02)*. ACM, New York, NY, USA, 2002, pp. 43–49. http://dx.doi.org/10.1145/581630.581638

[13] A. Farzan, and P. Madhusudan, "Causal Dataflow Analysis for Concurrent Programs," In: Orna Grumberg, Michael Hutch (eds.) *Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science*, vol. 4424. Proceedings of 13th International Conference, TACAS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007 Braga, Portugal, March 24–April 1, 2007, pp. 102–116, ISBN 978-3-540-71208-4. Berlin: Springer Berlin Heidelberg, 2007. [Online]. Available: http://www.cs.uiuc.edu/~madhu/tacas07.pdf.

[14] L. Bononi, M. Bracuto, G. D'Angelo, and L. Donatiello, "Concurrent Replication of Parallel and Distributed Simulations," In: *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation (PADS '05)*. IEEE Computer Society, Washington, DC, USA, 2005, pp. 234–243. http://dx.doi.org/10.1109/PADS.2005.6

[15] K. Huang, W. Haid, I. Bacivarov, M. Keller, and L. Thiele, "Embedding formal performance analysis into the design cycle of MPSoCs for real-time streaming applications," *ACM Trans. Embed. Comput. Syst.* 11, 1, Article no. 8 (April 2012), 2012, p. 23. http://dx.doi.org/10.1145/2146417.2146425

[16] M. Pradel, M. Huggler, and T. R. Gross, "Performance Regression Testing of Concurrent Classes", ISSTA'14, July 21–25, 2014, San Jose, CA, USA, pp. 13. http://dx.doi.org/10.1145/2610384.2610393

Vladislav Nazaruk was born in 1986. He received a degree of Bachelor of Engineering Science (Bc. sc. ing.) in Computer Control and Computer Science in 2007, and a degree of Master of Engineering Science (Mg. sc. ing.) in Computer Systems in 2009, both from Riga Technical University (RTU), Latvia. Now he is a Doctoral Student at the Institute of Applied Computer Systems, RTU.

He works as a Systems Analyst for *Ltd. ABC Software*. Fields of his research interests: computer science, mathematics, pedagogy. Special interests: algorithms, information theory, protection of information.

Address for correspondence: Meža Str. 1/3–506, Riga, LV-1007, Latvia;
E-mail: Vladislavs.Nazaruks@rtu.lv

Pavel Rusakov was born in Riga, Latvia in 1972. Dr. sc. ing. (1998), Mg. sc. ing. (1995), Bc. sc. ing. (1993) – Riga Technical University (RTU).

He is an Associated Professor at RTU, Institute of Applied Computer Systems. He is the Head of a Laboratory, responsible for the Professional Bachelor and Professional Master studies at the Department of Applied Computer Science. Field of interests: computer science. Special interests: programming paradigms, object-oriented approach to systems development, parallel computing, Web technologies, distributed systems, computer graphics, and protection of information.

Diploma with distinction: Mg. sc. ing.

Address for correspondence: Meža Str. 1/3–506, Riga, LV-1007, Latvia;
E-mail: Pavels.Rusakovs@cs.rtu.lv