

Analyzing Performance of Concurrent Usage Scenarios Using software Architecture Analysis

Tarja Kauppi and Anu Purhonen

VTT Electronics
P.O. Box 1100
90571 Oulu
Finland
Tel. +358 8 5512303

{Tarja.Kauppi, Anu.Purhonen}@vtt.fi

Abstract: This paper proposes an approach to performance analysis of a software architecture during concurrent usage scenarios and illustrates the use of the approach in practice with an industrial real-world case study. The approach combines three well-known concepts in the performance community, namely performance assessment of software architectures (PASA), layered queuing network (LQN), and rate monotonic analysis (RMA). PASA is an architecture analysis method that has evolved from software performance engineering (SPE). LQN is a performance modeling method and a notation that is an extension to queuing network modeling (QNM). Furthermore, RMA is a group of rules for analysis of the schedulability of a system. The goal is to provide software developers a systematic way in the early software system development phase to analyze the performance of concurrent usage scenarios. We have tried to approach the problem from the viewpoint of a developer that is not an expert in any particular performance modeling method so that the step to using more systematic methods would be as easy as possible.

Keywords: Software architecture, performance analysis

1. INTRODUCTION

Embedded system and software development has developed greatly since the first embedded products because of technological advanced in electronic design. For example, mobile phones have started to look like personal digital assistants (PDAs) instead of just phones. The users of the devices have greater freedom to use several applications simultaneously which compete for the same hardware resources. Even if the devices have considerable amount of resources available, they still have boundaries. Applications often require a real-time response, but the problem is, how to guarantee the quality of the service that the user has requested. One of the solutions for providing as good a service as possible for the user is to take into account performance concerns from the beginning of the development [19]. Architecture is the fundamental organization of a software system embodied in its components, their relationships to each other and to the environment [6]. Software architecture evaluation is a way of finding out problems in the early design when they are still easy to correct.

In our experience, performance analysis is not systematically used for supporting architectural design decisions in the industry. However, this does not mean that performance analysis is not used at all. On the contrary, support for design decisions has traditionally been gathered for example, using benchmarks, simulation, prototyping and analyzing of worst-case behavior based on experience. Some specific problems of the current approaches are that it is difficult to estimate the reliability of the results, the analysis cannot be easily repeated and the results are not comparable with other similar evaluations. In addition, usually each team or expert has their own methods and the results are not stored systematically so that they could be utilized in other evaluations and projects.

Since performance analysis has been studied for decades and several methods have already been applied to architecture evaluation [2, 4, 15, 23, 24, 26] the goal in this work was not to develop a new method but to use the current ones as far as possible. However, we were prepared to make some extensions to the existing methods if needed. To be accepted in the industry the method should be relatively easy and quick to apply, there should be enough instructions and case examples available concerning the method, and to speed up performance analysis there should be some tool supporting the method available.

PASA [24], a method for performance assessment of software architectures, is an evolution of the software performance engineering (SPE) [19] approach. Use of PASA is supported by two books [19, 21] and a commercial tool [20]. It has a clear definition of the steps from the architecture to the analysis and back. In that way it resembles the well-known software architecture evaluation methods that have been developed for other purposes [9, 10]. PASA (and SPE) supports the use of queuing network models (QNM) for performance modeling.

Queuing network models are a popular modeling method for performance analysis and they have been applied also to the software architecture analysis [7, 2, 1, 23, 3]. In our perspective, basic QNM have a drawback in that they do not support modeling of concurrent usage scenarios. Layered queuing network (LQN) models are an extension to QNM. LQN has been developed especially for concurrent and/or distributed software systems [17]. LQN has been applied to analyze unified modeling language (UML) [16] descriptions of high-level architectures and especially architectural patterns used in the system [17, 18].

Queuing networks-based methods are for analyzing average behavior. However, usually in the embedded system development the worst-case behavior is the more interesting one. Rate monotonic analysis (RMA) can be especially used for that purpose. RMA is a collection of rules to ensure that the system is schedulable [11, 15]. Based on our knowledge RMA rules on their own are difficult to use in large problems but, when the conditions of the analysis can be clearly defined and the focus of the analysis narrowed, they are helpful. The success of RMA often relies on how well the evaluator can select the correct rules.

More formal methods to performance analysis are Colored Petri Nets (CPN) [4, 26] and stochastic process algebras (SPA) [5]. However, from a normal software developer point of view they seem to be too laborious and time-consuming to construct. Moreover, SPA has been utilized to generate performance models from UML diagrams [12] but as far as we know there are not any examples where it would have been applied specifically to architecture analysis.

The problem of many of the current approaches is that they can usually be applied only in certain conditions that are not fulfilled in real world evaluation cases. For example, the worst-case behavior of a system occurs when

multiple usage scenarios are simultaneously active but, with current methods, it is often difficult to describe that kind of situation. In addition, in the early phases of the development life-cycle requirements usually change often and there is still much uncertainty about the values for the input parameters. Therefore, the performance models should be easy to change according to the changes in the architecture and input parameters. We have not wanted to overformalize the current development but to propose methods for the tasks where they were mostly needed and where the probability of taking them into use in the normal development process with average developers would be higher.

The paper is organized as follows. Section 2 presents our approach to analyzing the performance of concurrent usage scenarios and section 3 illustrates how the approach is used in practice. The results are discussed in Section 4 and Section 5 presents some concluding remarks.

2. METHOD OVERVIEW

Our performance evaluation approach is based on the PASA method [24], LQN modeling [25] and RMA principles [11]. The goals of the steps in the proposed approach are the same as in the PASA method, but LQN is used for performance modeling instead of QNM. RMA principles are used for finding better priorities for the tasks so that performance objectives are met, even in the worst-case condition. The steps of the approach are as follows:

1. The current or planned architecture is overviewed to obtain a high-level understanding of the architecture before going into its details.
2. Critical use cases are identified. They are the ones that are important to the operation of the system, or to responsiveness as seen by a user. They can also be the ones including significant performance risks, for example if not meeting performance objectives the system will fail or be less than successful.
3. Key performance scenarios are identified. The critical use cases identified in the previous step typically consist of a group of scenarios, which are not all important from the performance point of view. The key performance scenarios are the ones that are executed frequently and the ones that are critical to the user's perception of performance.
4. Performance objectives are identified for the analysis. At least one performance objective for each selected key performance scenario needs to be identified. Performance objectives have to be quantitative and measurable and they can be described in several ways including response time, throughput and constraints on resource usage. In addition the conditions under which the objectives should be achieved are defined.
5. The details of the architecture related to the selected key performance scenarios are clarified and discussed. The aim is to learn what architectural components relate to the problem area and how they interact with each other.
6. A LQN model [25] is drawn based on knowledge about the architecture. The model is analyzed either using the LQNS tool [25, 17] or by calculating some performance parameters such as utilization and residence time. Utilization means the average percent of time that the server (e.g. processor) is busy providing service and residence time means the average amount of time that jobs spend at the server [21]. The parameters can be calculated with the formulas (1) and (3) [21] if tasks are scheduled according to first-come-first-served or priority scheduling policy and if the system can be considered to be fast enough to handle arrivals so that throughput equals the arrival rate. If the conditions are not as assumed, then some other formulas need to be used instead. The formula for utilization is as follows:

$$U = XS, \quad (1)$$

where X is throughput and S is mean service time for jobs. Throughput means the average amount of time that jobs spend at the server and service time is the time spent receiving service from the resource [21]. An utilization upper limit for n tasks can be calculated with the following formula [11]:

$$U(n) = n(2^{1/n} - 1), \quad (2)$$

where n is the number of tasks. Residence time can be calculated with the following formula:

$$R = \frac{S}{1-U}, \quad (3)$$

where S is the time the server is busy providing service and U is utilization percent of the server. Residence time can be compared to identified performance objectives.

7. If the previous steps point out that performance objectives are not met, then some changes are needed to the system to meet the objectives. One way to improve performance may be to modify the software architecture by using performance antipatterns [21] as an aid. They are documented bad design decisions from the performance point of view, including also solutions to remove them. Another way may be to change the

priorities of the tasks according to RMA scheduling principles [11]. If calculated utilization is less than the utilization limit and if it can be assumed that the tasks are periodic, are not synchronized with another, do not suspend themselves during execution, and are capable of being preempted by higher priority tasks [11, 21], then priorities can be assigned in a way that performance objectives are met even in the worst case situation. In the mentioned situation the task with highest arrival rate should be assigned the highest priority, the task with the next highest priority should be assigned the next highest priority and so on. This way tasks should always meet their deadlines.

3. EXAMPLE ANALYSIS

A mobile phone system is a good example of a highly complex system providing many kind of services to its users and many of the services requiring real-time response. The aim of this case study was to analyze the software architecture to find out if the software system is able to simultaneously process continuous multimedia streaming and multimedia message (MMS) and still provide the required quality of service. According to the 3GPP streaming service standard [27] streaming is the ability of an application to play media streams like audio and video in a continuous way while those streams are being transmitted to the client over a data network. So, received multimedia streaming content should be handled and rendered in time to look and sound continuous from the user point of view. Multimedia messages can include multiple media such as audio, video, text and picture [14]. Receiving a large MMS during streaming may degrade the quality of the streaming service.

This case study is based on an industrial real world planned architecture. However, some of the details have been modified in this paper to preserve confidentiality. The analysis was conducted according to the approach presented in the previous chapter and based on software architectural and functional design documentation and estimated execution times. The case study can be found with more details from [8].

3.1 Architecture Description

Architecture Overview

First was examined that the high-level target software architecture comprises of three components: Symbian software, DSP software and bearer protocols software, each of which is deployed to different processors. In this case study, we concentrated only on analyzing the performance of the Symbian SW component, which is running on the Symbian operating system [22].

Critical Use Cases

Streaming and receiving an MMS message were identified to be the critical use cases. Streaming is a time-critical use case and the MMS message was seen to be the most likely to cause performance problems for streaming.

Key Performance Scenarios

Streaming consists of several scenarios such as establishing a session to the streaming content provider, tearing down the session and receiving multimedia stream packets from the content provider [27]. The two first scenarios are executed only once, but stream packets are received frequently. Delays related to handling and rendering stream packets can be noticed by the user. Consequently, receiving stream packets (audio and video) were selected to be the key performance scenarios. Receiving an MMS message consists of two separate scenarios: receiving a notification of the message and receiving MMS content packets. The notification is received only once per one MMS message and therefore the impact that it has on the performance of streaming was considered to be insignificant. The actual content of the MMS message consists of multiple packets and the competition for the same resources with streaming may cause performance problems. Therefore, receiving MMS content packets was also selected to be one of the key performance scenarios.

Performance Objectives

The streaming requires almost real-time behavior, but receiving an MMS message does not have real-time requirements. An interesting condition from the performance point of view was considered to be when stream data is received with a high packet arrival rate and in relatively small packets. The values mentioned were in this case 40 packets/s 200 bytes/packet for video stream and 10 packets/s 150 bytes/packet for audio stream. The maximum MMS message size is 100 Kbytes [13] and it was assumed in this case that the message is received in 100 packets with 1 packet/s arrival rate.

It was assumed that the objective is that only one of each packet type at maximum is in the Symbian software. Based on the assumed arrival rates and the mentioned objective it was concluded that the maximum allowed response time for video packets is 25 ms and for audio packets 100 ms. The defined performance objectives should also be met when an MMS message is received concurrently. Even if receiving an MMS message does not require real-time response the number of MMS content packets at the processor effects the performance of streaming. Therefore, it was defined that there should not be more than one MMS content packet at the processor. Hence, the maximum response time for an MMS content packet is 1000 ms.

Architectural details

Some of the architectural details were not clarified based on the available software architectural and functional design documentation. These kinds of issues were discussed in technical meetings with the customer organization. For example, the communication between different software components and software deployment to the hardware needed clarification. Each message between the software components related to streaming and MMS was assumed to be sent once per one arrived packet and they were assumed not to wait for a reply message. The software architecture components related to the problem area are presented in an execution graph format [21] in Figure 1, where rectangles present the related software components and arcs denote the execution order of the components. The rectangles with a solid line belong to the problem area analyzed in this case study. The numbers next to the rectangles are estimated execution times required from the processor. The estimations are based on measurements with the same scenarios in different kind of system. The measured values were scaled to correspond the values in the target system.

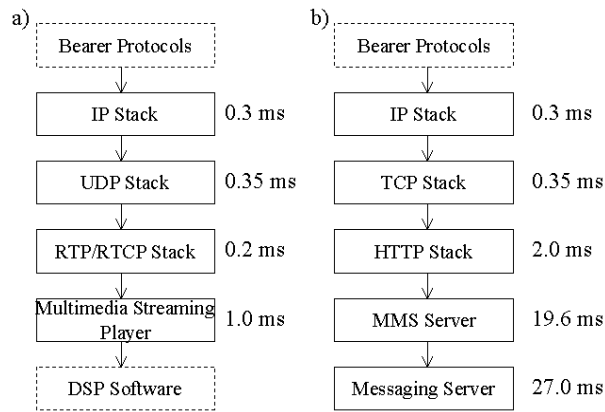


Figure 1. Execution graphs for a) streaming and b) MMS content packet reception.

3.2 Analysis

After finding out the required information about the architecture and execution times the LQN model (Figure 2) was drawn by applying the instructions given in the LQN tutorial [25]. No tool support was available and therefore the model was analyzed by calculating utilization percentage of the processor with formula (1) and average residence times of different tasks with formula (3). In this case study the task was assumed to be a packet (video stream, audio stream or MMS content) and the packet was assumed not to fall to pieces in the data path. Utilization bound for three tasks was calculated with formula (2). The calculated values and corresponding maximum allowed values are presented in Table 1.

Table 1. Performance metrics

Performance Metric	Value	Maximum allowed value
Utilization (U)	14.2 %	77.9 %
Residence time (R) for video stream packet	2.16 ms	25 ms
Residence time (R) for audio stream packet	2.16 ms	100 ms
Residence time (R) for MMS content packet	57.38 ms	1000 ms

The formulas used were applicable, because the system could be assumed to act like a priority scheduled system. The utilization was noticed to be less than the upper limit for utilization. By comparing the calculated residence times and identified response time objectives it was noticed that on average the performance objectives are met. Based on the calculations it could be concluded that, if the execution times are as assumed on average, there

should not be performance problems from the processor point of view. So, changes were not necessarily required to the system, but meeting performance objectives in a worst-case condition could still not be guaranteed.

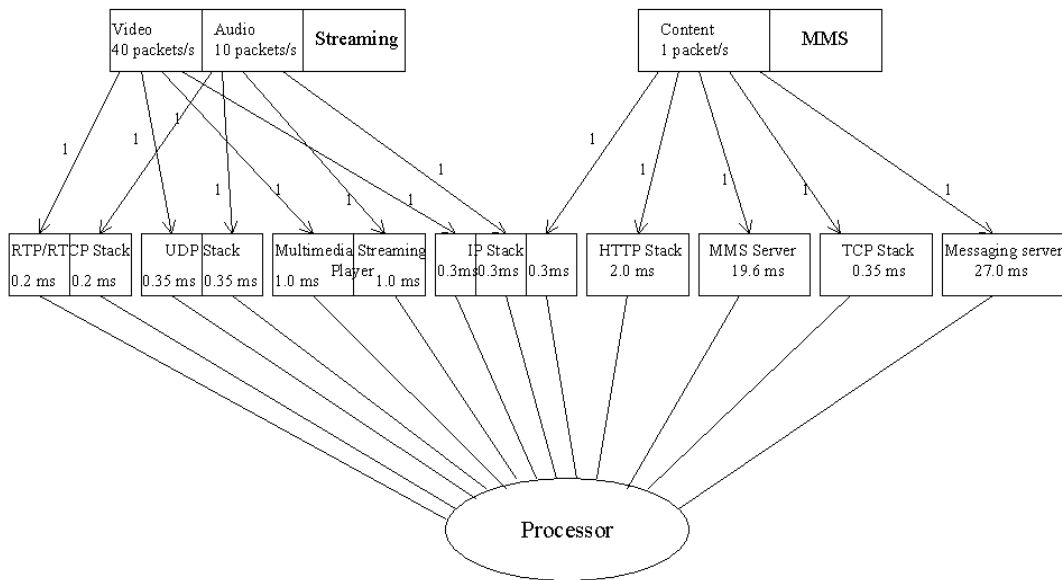


Figure 2. The LQN model.

3.3 Alternatives

Because improvements were not necessarily needed to the software architecture, performance antipatterns were not used. Tasks were assumed to be periodic, not to be synchronized with another, not to suspend themselves during execution and to be capable of being preempted by a higher priority task and utilization is less than maximum utilization limit, therefore it could be concluded that the system is schedulable. In this case video packets have the highest arrival rate, audio packets have the second highest and MMS content packets have the lowest arrival rate. Therefore, by assigning video packets the highest priority, audio packets the second highest priority and MMS the third highest priority, the tasks can be guaranteed to meet their deadlines.

4. DISCUSSION

Our primary purpose was to find a method for analyzing the performance of a software architecture during concurrently processed scenarios and illustrate the application of the method with an industrial real-world case example. We started by studying the suitability of existing software architecture-based performance analysis methods. The PASA method was seen to be most suitable, however it needed some extensions to be applicable. Our approach uses PASA as an overall evaluation framework, LQN for performance modeling and RMA principles for finding suitable priorities. As a result of this work software architecture-based performance analysis was noticed to provide added value to the software system development. Even if the quantitative values produced are approximations, the presented approach helps to manage the performance of a complex system, because it guides to concentration on the parts of the software architecture that are important from the performance point of view.

Based on this research at least three problematic issues related to early performance analysis were found which may effect the reliability of the results. The first problem is does the performance model illustrate the system in enough detail. Second, where to obtain the needed timing values if nothing can be measured. Third, how do the made assumptions effect on the results.

There are some limitations on this study. First, the time for the analysis case was limited and therefore, the problem area had to be outlined to be able to perform the work in the given time. For example the processor was the only hardware resource that was considered. A second limitation of the present study is that the evaluator had no earlier experience of the target system or applying performance analysis methods in practice. However, this was not a problem, because the aim was to approach the problem as an ordinary developer with no experience

with performance analysis methods. Third, no tool supporting the applied methods was available. Therefore, performance parameters had to be calculated with self-chosen formulas.

The approach probably produces the best results if the same type of system that is under development has earlier been implemented. In that case the required timing information can be estimated based on earlier measurements. If the type of system is under development for the first time, then quantitative analysis may not be meaningful, because obtaining accurate enough timing information is difficult, or even impossible. However, in that case the software architecture designer can use performance anti-patterns as an aid, because they do not require quantitative information. This may help the designer to avoid design decisions that have a negative impact on performance. It should also be noticed that the formulas used in this study are applicable only if the system acts like a first-come-first served or priority scheduled system. Otherwise some other formulas need to be used.

The reliability of the results from early performance analysis should be somehow proved in the future, for example by comparing measured results and the values obtained from performance analysis. Then the acceptance from the industrial perspective would be higher. Also the usage of available tools related to early performance analysis should be explored to find out what kind of constraints there are (e.g. scheduling policies, arrival patterns, etc.), what kind of inputs they require and what kind of outputs they produce. Also it would be useful to find some kind of support for estimating timing parameters in the early development phase.

5. CONCLUSION

This paper presented an approach to analyzing performance of concurrent usage scenarios using software architecture evaluation. The approach combines three well-known methods PASA, LQN and RMA. PASA gives the overall framework for the evaluation, LQN is used for performance modeling and RMA for finding priorities so that deadlines are met even in the worst-case situation. The purpose of the method is to support the development of software products when the evaluators are not experts in queuing theory or other paradigms. Although there is still much to do the results show that this approach can be useful in analyzing real world software architectures.

Acknowledgements

This work has been partially conducted in MOOSE (“Software engineering methodologies for embedded systems”) project under the ITEA cluster project of the EUREKA network, and financially supported by TEKES (Technology Development Center of Finland). Anu Purhonen was also supported by a grant from the Nokia Foundation.

References

- [1] F. Andolfi, F. Aquilani, S. Balsamo, P. Inverardi. 2000. Deriving Performance Models of Software Architectures from Message Sequence Charts. In Proceedings of WOSP 2000: Second International Workshop on Software and Performance, Ottawa, Canada, September 17-20, 2000. ACM. Pp. 47-57.
- [2] F. Aquilani, S. Balsamo, P. Inverardi. 2001. Performance analysis at the software architectural design level. Performance Evaluation. Vol. 45.No. 2-3. Pp. 147-178.
- [3] S. Balsamo, V. De Nitto Personè, P. Inverardi. 2003. A review on queuing network models with finite capacity queues for software architectures performance prediction. Performance Evaluation. Vol. 51, Nos. 2-4. Pp. 269-288.
- [4] K. Fukuzawa, M. Saeki. 2002. Evaluating Software Architectures by Coloured Petri Nets. In Proceedings of the 14th International Conferences on Software Engineering and Knowledge Engineering, SEKE’02, July 2002, Ischia, Italy. Pp. 263-270.
- [5] U. Herzog, J. Rolia. 2001. Performance validation tools for software/hardware systems. Performance Evaluation. Vol. 45, No. 2-3. Pp. 125-146.
- [6] IEEE, IEEE Recommended Practice for Architectural Description of Software-Intensive Systems, IEEE Std 1471-2000.
- [7] P. Inverardi, A.L. Wolf. 1995. Formal Specification and Analysis of Software Architectures Using the Chemical Abstract Machine Model. IEEE Transaction on Software Engineering. Vol. 21, No. 4. Pp. 373-386.

- [8] T. Kauppi. 2003. Performance Analysis at the Software Architectural Level. University of Oulu, Department of Electrical Engineering. Diploma Thesis, 56 p.
- [9] R. Kazman, G. Abowd, L. Bass, P. Clements. 1996. Scenario-Based Analysis of Software Architecture. *IEEE Software*. Vol. 13, No. 6. Pp. 47-55.
- [10] R. Kazman, M. Klein, M. Barbacci, H. Lipson, T. Longstaff, S. J. Carrière. 1998. The Architecture Tradeoff Analysis Method. *Proceedings of ICECCS, Monterey, CA, August 1998*. Pp. 68-78.
- [11] M.H. Klein, T. Ralya, B. Pollak, R. Obenza, M.G. Harbour. 1993. *A Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*. Massachusetts: Kluwer Academic Publishers.
- [12] C. Lindemann, A. Thümmler, A. Klemm, M. Lohmann, O.P. Waldhorst. 2002. Performance analysis of time-enhanced UML diagrams based on stochastic processes. In *Proceedings of the Third International Workshop on Software and Performance (WOSP'02), Rome, Italy, ACM (July 2002)*. Pp. 25-34.
- [13] MMS: Multimedia Messaging Service FAQ. 2003. Canvas Dreams. Beaverton. United States
URL: <http://www.canvasdreams.com/viewarticle.cfm?articleid=1186>.
- [14] Nokia. 2002. Nokia Multimedia Messaging: Show what you mean. Nokia White Paper. URL: <http://nds1.nokia.com/press/pdfs/mmsA4.pdf>.
- [15] R.L. Nord, B.C. Cheng. 1994. Using RMA in Evaluating Design Decisions, Position paper, *Proceedings of the Second IEEE Workshop on Real-Time Applications*, IEEE Computer Society, Washington D.C. USA, July 1994.
- [16] Object Management Group, OMG Unified Modeling Language Specification, Version 1.5, <http://www.uml.org>.
- [17] D. Petriu, C. Shousha, A. Jalnapurkar. 2000. Architecture-Based Performance Analysis Applied to a Telecommunication System, *IEEE Transactions on Software Engineering*, Vol. 26, No. 11. Pp. 1049-1065.
- [18] D. Petriu, X. Wang. 1999. From UML descriptions of High-Level Software Architectures to LQN Performance Models. In *Proceedings of AGTIVE'99*, Springer Verlag, LNCS 1779, Pp. 47-62.
- [19] C.U. Smith. 1990. *Performance Engineering of Software Systems*. Addison Wesley. 570 p.
- [20] C.U. Smith, L.G. Williams. 1997. Performance Engineering Evaluation of Object Oriented Systems with SPE•ED in *Computer Performance Evaluation: Modelling Techniques and Tools*. LNCS 1245. (R. Marie et al. Eds.) Springer Verlag. Pp. 135-154.
- [21] C.U. Smith, L.G. Williams. 2002. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley, Boston, 510 p.
- [22] Symbian OS - the mobile operating system. 2003. Symbian Ltd. URL: <http://www.symbian.com>.
- [23] B. Spitznagel, D. Garlan. 1998. Architecture-Based Performance Analysis. In *Proceedings of the 1998 Conference on Software Engineering and Knowledge Engineering (SEKE'98)*, June 1998.
- [24] L.G. Williams, C.U. Smith. 2002. PASA: A method for the Performance Assessment of Software Architectures. In: *Proceedings of the Third International Workshop on Software and Performance (WOSP '2002)*, July 24-26, Rome, Italy. Pp. 179-189.
- [25] M. Woodside. 2003. Tutorial Introduction to Layered Modeling of Software Performance. Edition 3.0. Carleton University. URL: <http://www.sce.carleton.ca/rads/lqn/lqn-documentation/tutorialf.pdf>.
- [26] J. Xu, J. Kuusela. 1998. Analyzing the execution architecture of mobile phone software with colored Petri nets, *International Journal on Software Tools for Technology Transfer*, Vol. 2 No 2. Pp. 133-143.
- [27] 3GPP TS 26.233 V4.2.0. 2002. Transparent end-to-end packet switched streaming service (PSS): General description. 3rd Generation Partnership Project, Technical Specification Group Services and System Aspects, Release 4, Sophia Antipolis, France, 11 p.