Phymss – Performance Hybrid Model Solver and Simulator Based on UML MARTE Diagrams

Cosmina Chişe Ioan Jurca

* Department of Computer Science and Engineering,
"Politehnica" University of Timisoara, Faculty of Automation and Computers,
2 V. Parvan Blvd, Timişoara, Romania
0040745024679

{chise.cosmina, ioan.jurca}@gmail.com

ABSTRACT

There are several research directions in Software Performance Engineering (SPE), covering the entire performance prediction process, but most of the tools developed so far implement only part of it or have restrictions. From a methodology perspective, current performance prediction tools rely either on analytical or simulation models, as separate techniques. This paper presents a performance analysis tool, Phymss (Performace Hybrid Model Solver and Simulator), which covers the analysis process from the input system model annotated with performance information to obtaining performance results and inserting them back into the original system model. Two analysis methods are implemented, for flexibility reasons: a multithreaded simulator and a hybrid solver that combines the analytical and simulation approaches in a new analysis technique, in order to investigate the benefits of such an approach.

Categories and Subject Descriptors

D.2.2 [Software Engineering]: Design Tools and Techniques – c*omputer-aided software engineering (CASE).*

General Terms

Performance, Design.

1. INTRODUCTION

There are several research directions in Software Performance Engineering (SPE), covering the entire performance prediction process, from input language, model extraction, to model solvers or simulators. Performance prediction tools rely on analytical or simulation models. Analytical approaches are fast, but not accurate and cannot be applied to systems with complex behavior. Simulation models can be derived even from complex systems, the drawback being the large number of iterations that need to be performed in order to obtain relevant mean values for parameters.

Hybrid approaches exist for network processor design [1], but the methods are separately defined, have distinct input and output parameters and they are only ordered one after the other. Another has been used to combine software design evaluation with network specific architecture in [7]. The software model (Layered Queueing Network) and network model (NS-2 [8]) are solved iteratively, thus the analysis results are refined.

This paper introduces Phymss (Performance Hybrid Model Solver and Simulator), a tool that intends to encompass as much as possible from the performance analysis process, based on an improved hybrid meta-model. Both a simulator and a hybrid solver are available for performance analysis.

WOSP/SIPEW'10, January 28–30, 2010, San Jose, California, USA.
ACM 978-1-60558-563-5/10/01.

2. PHYMSS TOOL

Phymss implements two performance analysis techniques: a simulation approach and a hybrid method. The tool is developed in C#, using Microsoft .NET Framework 3.5. The block diagram of the system is presented in Figure 1.

Tool input is represented in XMI (XML Metadata Interchange) format and can be obtained as output from visual design editors for UML (Unified Modeling Language) diagrams, such as Papyrus UML [9], which supports extensions for the MARTE (Modeling and Analysis of Real Time and Embedded systems) profile. Simulation and analytical solving parameters, such as duration or iteration count are specified in a JavaScript configuration file; this language has been chosen, in order to be easily adopted by users and interpreted by the application. This configuration file is also useful in order to parameterize the system model description – variables can be left unassigned inside the XMI file, and their values specified in the configuration file. Hence, the effects of different values for system parameters on system performance can be evaluated without changing the UML model, only the configuration file needs to be changed.

Inside the system, which is illustrated as the darker-shaded box, the UML model is stored with all performance annotations; this is where performance results are stored too, during simulation or hybrid evaluation, as shown by the two highlighted alternative paths.

After having applied one of the two approaches, performance analysis results can be exported into an XMI file, with the same structure as the input file: each UML node will have values specified for parameters such as response time, throughput or utilization.

The pure simulator builds the simulation model from the UML model and executes it, inserting the results into the UML model as statistics while the simulation runs. It is based on the simulation model defined by Marzolla in UML-Ψ [4]. The implementation does not rely on single-threaded coroutines, as in the original approach, but is improved by using thread pools, and thus allowing for multiple threads to be run simultaneously. Considering that a dual-core processor is rather usual in deployed systems, multithreaded simulation for models of such systems will provide more accurate results than a single-threaded one.

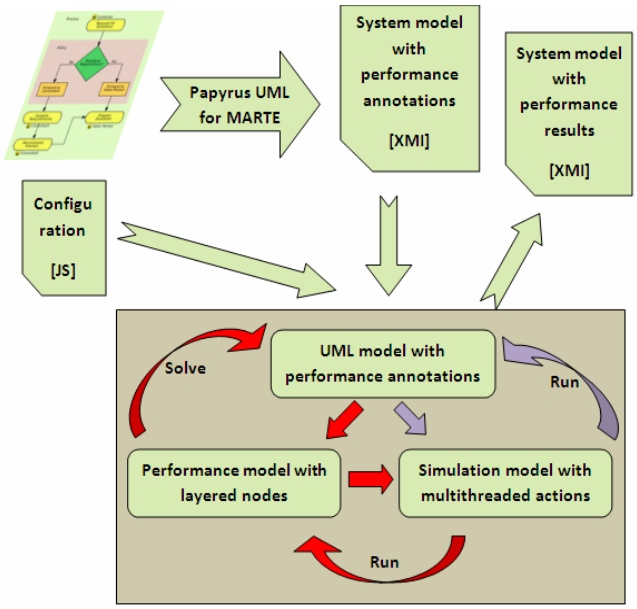

Figure 1. Phymss block diagram

The hybrid approach requires building the performance model, which in turn creates a simulation submodel that is run. The approach is based on the hierarchical decomposition of the system model into submodels, used by LQNS (Layered Queueing Network Solver) [3]. Each submodel is solved as a Closed Queueing Network (CQN) by Mean Value Analysis (MVA) or Approximate MVA [5], the results being propagated among levels. Franks proves in [3] that there is a two-way dependency between levels, so performance parameter values need to be propagated both downwards and upwards. The simulation step is inserted between the two analytical passes, starting from a level k; the simulation submodel includes items on level $k+1$ and all lower levels. In this case, an iteration consists of three steps (Figure 2): submodels are solved starting from the highest level, in order to propagate think time values along the request chains until level k is reached; level k and lower levels are simulated; submodels are solved starting from level k back to level 1, to propagate upwards service time values.

$Cummulate(S_1)$ // add service time values for all servers
DO iteration
  $Generate(Z_1)$ // generate think time values for requests
  FOR submodel $l = 1..k$
    $MVA(l, Z_l, S_l) \Rightarrow \rho_l, \lambda_l$ // utilization, throughput
    $Z_{l+1} = f(\rho_l, \lambda_l)$ // propagate think time values
  $Simulate(M_{k+1}, Z_{k+1}) \Rightarrow P_{k+1}, \Lambda_{k+1}, T_{k+1}$ // utilization,
                                                // throughput, response time
  FOR submodel $l = k..1$
    $S_l = f(T_{l+1})$ // propagate service time values
    $MVA(l, Z_l, S_l) \Rightarrow \rho_l', \lambda_l'$
WHILE (NOT $Convergent(P_1, \Lambda_1, T_1)$ AND iteration count not exceeded)

Figure 2. Pseudo-code for hybrid solver

The method relies on a performance metamodel that can easily be extracted from system specifications, expressed using the MARTE profile. This meta-model has been defined in [2] and is based on UML-Ψ [4] and Core Scenario Model (CSM) [6]. An advantage of this new metamodel is its simplicity, eliminating redundant elements, while maintaining a clear structure for system components and their interactions.

Because of the approximations used in formulae during LQN solving, the performance results are not as accurate as simulation results, but this approach is obviously faster than the simulator. Regarding improvements in results accuracy, better approximations are to be implemented, since this is the first attempt in implementing MVA.

The input model range that can be analyzed by the hybrid approach will be extended from CQN to QNs that have mixed request types, both open and closed. An analytical approach regarding models accepting mixed requests is already available in [5]: the open queueing network is solved first and the results are used to "inflate" the service times of tasks that accept both closed and open requests, and then the resulting CQN is solved.

The hybrid method can further be improved by establishing an appropriate level k, from which simulation should be performed. Given the total number of layers and their complexity, k can be heuristically computed depending on each analyzed system.

3. REFERENCES

[1] Chakraborty, S., Kunzli, S., Thiele, L., Herkersdorf, A., and Sagmeister, P. 2003. Performance evaluation of network processor architectures: combining simulation with analytical estimation. Elsevier Science B.V.

[2] Chişe, C., and Jurca, I. 2009. Towards Early Performance Assessment Based on UML MARTE Models for Distributed Systems. In Proc. of the SACI (Timisoara, Romania, May 2009)

[3] Franks, G., 1999 Performance Analysis of Distributed Server Systems. Doctoral Thesis. Department of Systems and Computer Engineering, Carleton University, Ottawa, Canada

[4] Marzolla, M., and Balsamo, S. 2004. UML-PSI: The UML Performance Simulator. In Proc. of the 1st Int. Conf. on Quantitative Evaluation of Systems (Enschede, The Netherlands, September 27-30, 2004). QEST 2004. pp. 340-341.

[5] Munoz, L., Lecture on Computer Design and Evaluation. Mean value Analysis, Universidad Politecnica de Madrid, http://www.datsi.fi.upm.es/docencia/DEC

[6] Petriu, D. B., and Woodside, M. 2004. A metamodel for generating performance models from UML designs. In Proc. of the UML 2004 conference, vol. 3273 of Lecture Notes in Computer Science (LNCS 3273), pp. 41-53 (Lisbon, October 2004) ftp://ftp.sce.carleton.ca/pub/cmw/csm-uml04.pdf

[7] Verdickt, T., De Turck, F., Dhoedt, B., and Demeester, P. 2007. Hybrid performance modeling approach for network intensive distributed software. In Proc. of the 6th Int. Workshop on Software and Performance (Buenos Aires, Argentina, February 5-8, 2007). WOSP 2007.

[8] NS-2 network simulation tool, Internet, http://www.isi.edu/nsnam/ns/

[9] Papyrus UML tool, Internet, http://www.papyrusuml.org/