Performance-Driven Software Architecture Refactoring

Davide Arcelli, Vittorio Cortellessa, and Daniele Di Pompeo
Department of Information Engineering, Computer Science and Mathematics
University of L'Aquila, Italy
{*davide.arcelli, vittorio.cortellessa*}*@univaq.it,*
daniele.dipompeo@graduate.univaq.it

***Abstract*—Performance engineering of software architecture can be defined as the process of analyzing the performance of a software architecture and then reacting to problems emerging from such analysis by refactoring the software architecture in order to meet performance requirements. In the last decade, many approaches in this field have appeared, whereas the problem of reacting to problems by proposing and evaluating alternative solutions through architectural refactoring has been much less treated. Indeed, the introduction of automated support to refactoring becomes crucial to drive architectural evolutions that might lead to performance improvement. This tutorial is aimed at introducing notations, methodologies and tools that can be adopted for Performance-Driven Software Architecture Refactoring.**

***Keywords*-Software Refactoring; Software Performance Engineering; Software Architecture**

I. The tutorial topic

Performance-Driven Software Architecture Refactoring (PDSAR) can be defined as a discipline that collects approaches, methodologies, and tools aimed at introducing performance assessment in the context of software architectures. Figure 1 depicts a PDSAR process and it represents the context addressed by this tutorial. The process has three horizontal swimlanes, namely *Forward path*, *Backward path* and *Knowledge*. The latter contains system and domain artifacts (e.g. *Architecture Under Analysis* – AUA – and *Refactoring Library*) supporting the process.

The forward path starts with the *SAM2PM* model-to-model transformation (e.g. [1]) aimed at generating a performance model from a software architecture. This step is followed by a *Performance Analysis* procedure (e.g. [2]) that carries out the *Performance Values of Interest* corresponding to the metrics specified in the knowledge repository.

The obtained performance values are then taken as input by the first step of the backward path, namely *Metric values integration*, which properly fills the AUA with the metric values, thus obtaining an *Annotated AUA*. The latter undergoes a *Results interpretation* step that uses the knowledge needed to detect performance problems (e.g., [3]).

The *Results Interpretation* step produces a list of *Performance Problems Occurrences*. Based on the latter, an *Architectural feedback generation* step is then executed, with the support of *Refactoring Actions* knowledge repository. The result of this step is represented by a set of *Architectural Alternatives* that go back to the forward path in order to be evaluated. *Architectural Alternatives* that satisfy performance requirements and/or that show better performance than the AUA can thus finally be considered as beneficial evolutions of the latter.

The whole process is plugged into an *Architectural Notations* frame in Figure 1, which accounts for the different notations that can be used in different instantiation of the process. The focus of this tutorial is the backward path and, consequently, the knowledge supporting it.

II. State of art

The quality improvement of software architectures is a problem known to be computationally hard, due to the typically huge space of feasible solutions, complexity of architectural notations, heterogeneity/fragmentation of available methods and tools.In the performance domain, existing approaches to this problem mostly rely on heuristics and metaheuristics, while addressing particular architectural notations (as for, e.g., UML-MARTE [4] and AADL [5]) that often natively support performance analysis (e.g. PCM [6] and Æmilia [7]).

Heuristics that exploit performance antipatterns to specify performance problems at the architectural level are of particular interest, since performance antipatterns are patterns characterising bad design practices that likely jeopardize performance [8]. However, most of them are limited to automation of antipatterns detection [3], [9], [6], [7], whereas there is lack of exploitation of Model-Driven Engineering techniques to introduce automated support for planning and applying architectural refactoring actions, possibly based on antipattern/bottleneck removal [10], [11], [12], [13].

Moreover, a number of more recent studies have demonstrated the effectiveness of formulating the problem as a search-based one, and of tackling it via metaheuristics [14]. Several evolutionary algorithms have been introduced, in the last decade, for software architecture multi-objective optimization with respect to various quality attributes, including performance [15], and with different degrees of freedom to modify the architecture [16], [17].

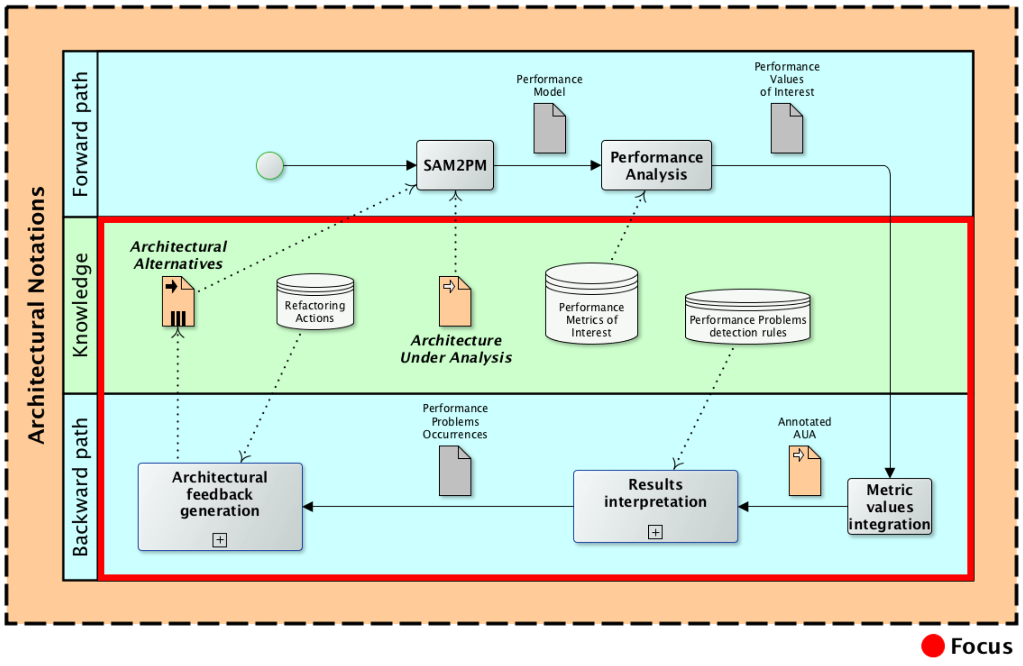
Architectural Notations
Forward path
Knowledge
Backward path
Performance Model
Performance Values of Interest
SAM2PM
Performance Analysis
Architectural Alternatives
Refactoring Actions
Architecture Under Analysis
Performance Metrics of Interest
Performance Problems detection rules
Performance Problems Occurrences
Annotated AUA
Architectural feedback generation
Results interpretation
Metric values integration
Focus

Figure 1. Context and focus of this tutorial.

III. Target audience

The potential attendees of this tutorial are researchers and practitioners that, respectively, are interested to investigate issues related to this topic and experience performance problems in their working activity. They will be exposed to MDE techniques that can be adopted to introduce automated support to software architecture refactoring driven by performance. They will also be exposed to the insight that automation in this process lowers the bar of its applicability, thus paving the way to the development of solutions that actually support the performance-driven refactoring.

The basic prerequisites for this tutorial are: software architecture, basic concepts of performance analysis, model-driven engineering basics, and pattern/antipattern concepts.

References

[1] C. M. Woodside, D. C. Petriu, J. Merseguer, D. B. Petriu, and M. Alhaj, “Transformation challenges: from software models to performance models,” *Software and System Modeling*, vol. 13, no. 4, pp. 1529–1552, 2014.

[2] G. Casale and G. Serazzi, “Quantitative system evaluation with java modeling tools,” in *ICPE’11*, 2011, pp. 449–454.

[3] V. Cortellessa, A. Di Marco, and C. Trubiani, “An approach for modeling and detecting software performance antipatterns based on first-order logics.” *Software and System Modeling*, vol. 13, no. 1, pp. 391–432, 2014.

[4] D. Arcelli, V. Cortellessa, and C. Trubiani, “Antipattern-based model refactoring for software performance improvement,” in *QoSA’11*, 2011, pp. 33–42.

[5] A. Aleti, S. Björnander, L. Grunske, and I. Meedeniya, “Archeopterix: An extendable tool for architecture optimization of AADL models,” in *MOMPES (ICSE’09)*, pp. 61–71.

[6] C. Trubiani and A. Koziolek, “Detection and solution of software performance antipatterns in palladio architectural models,” in *ICPE’11*, 2011, pp. 19–30.

[7] M. De Sanctis, C. Trubiani, V. Cortellessa, A. Di Marco, and M. Flamminj, “A model-driven approach to catch performance antipatterns in ADL specifications,” *Information & Software Technology*, vol. 83, pp. 35–54, 2017.

[8] C. U. Smith and L. G. Williams, “Software performance antipatterns for identifying and correcting performance problems,” in *ICMG’12*, 2012.

[9] A. Wert, M. Oehler, C. Heger, and R. Farahbod, “Automatic detection of performance anti-patterns in inter-component communications,” in *QoSA’14*, 2014, pp. 3–12.

[10] J. Xu, “Rule-based automatic software performance diagnosis and improvement,” *Performance Evaluation Journal*, vol. 69, no. 11, pp. 525–550, 2008.

[11] A. Amirat, A. Bouchouk, M. O. Yeslem, and N. Gasmallah, “Refactor software architecture using graph transformation approach,” *INTECH’12*, pp. 117–122, 2012.

[12] D. Arcelli, V. Cortellessa, and D. D. Ruscio, “Applying model differences to automate performance-driven refactoring of software models,” in *EPEW’13*, 2013, pp. 312–324.

[13] D. Arcelli, V. Cortellessa, D. D. Pompeo, “Performance-driven software model refactoring,” *Information & Software Technology*, vol. 95, pp. 366–397, 2018.

[14] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, “Software Architecture Optimization Methods: A Systematic Literature Review,” *IEEE Trans. on Software Engineering*, vol. 39, no. 5, pp. 658–683, 2013.

[15] A. Koziolek, H. Koziolek, and R. Reussner, “PerOpteryx: automated application of tactics in multi-objective software architecture optimization,” in *QoSA’11*, 2011, pp. 33–42.

[16] F. Rosenberg, M. B. Müller, P. Leitner, A. Michlmayr, A. Bouguettaya, and S. Dustdar, “Metaheuristic optimization of large-scale qos-aware service compositions,” in *SCC’10*, 2010, pp. 97–104.

[17] V. Cardellini, E. Casalicchio, V. Grassi, F. Lo Presti, and R. Mirandola, “Qos-driven runtime adaptation of service oriented architectures,” in *ESEC/FSE’09*. ACM, 2009, pp. 131–140.