The Vision of Self-Aware Performance Models

Johannes Grohmann
University of Würzburg
Würzburg, Germany
johannes.grohmann@uni-wuerzburg.de

Simon Eismann
University of Würzburg
Würzburg, Germany
simon.eismann@uni-wuerzburg.de

Samuel Kounev
University of Würzburg
Würzburg, Germany
samuel.kounev@uni-wuerzburg.de

***Abstract*—Performance models are necessary components of self-aware computing systems, as they allow such systems to reason about their own state and behavior. Research in this field has developed a multitude of approaches to create, maintain, and solve performance models. In this paper, we propose a meta-self-aware computing approach making the processes of model creation, maintenance and solution themselves self-aware. This enables the automated selection and adaption of software performance engineering approaches specifically tailored to the system under study.**

I. Introduction

Self-aware computing systems are defined as computing systems that: (1) learn models capturing knowledge about themselves and their environment on an ongoing basis and (2) reason using the models enabling them to act based on their knowledge and reasoning in accordance with higher-level goals, which may also be subject to change [1].

Self-aware data centers are one application domain for self-aware computing systems. The data center collects knowledge about itself in the form of models and then uses these models to reason about its own state and behavior. For example, performance concerns of the applications hosted in the data center can be addressed using performance models such as queueing networks or architecture-level performance models like the Descartes Modeling Language (DML) [2]. This enables, e.g., bottleneck analysis [3], Service Level Agreement (SLA) evaluation [4], proactive auto-scaling [5] or configuration optimization [6]. These performance models serve as the knowledge base or the model representation of the model-based learning and reasoning loop (LRA-M loop) of the self-aware system [1]. Several approaches for the extraction of performance models exist (e.g., [7], [8], [9], [10]), which can be used to extract a performance model using monitoring data of a software system.

Our vision is to introduce the concept of meta-reflective self-awareness or simply meta-self-awareness [11] in the area of performance modelling, by making the model learning and prediction processes themselves self-aware. The self-aware model learning and prediction processes can reason about their own performance and capabilities and based on this reasoning adapt themselves at run-time. In order to do so, we identify four areas involved in the performance model extraction and prediction processes that can be improved by introducing meta-self-awareness.

The main benefit of self-awareness is that the system learns which algorithms work best for each specific problem instance. The system automatically adapts to the current situation by selecting the best suited algorithms for the specific setting. In the following, we detail how classical performance models can be enhanced by adding self-aware capabilities, enabling the construction of systems with meta-self-awareness.

II. Overview

Our approach as shown in Figure 1 is composed of four interrelated steps: (1) query-based model tailoring, (2) query-tailored model solution, (3) model validation and (4) model recalibration. Two repositories serve as knowledge base, storing the model history (model repository) as well as measurement and prediction history (performance repository). Questions about the performance behavior of the system can be formulated as so called performance queries, specifying the performance metric of interest, the required accuracy and an upper time limit.

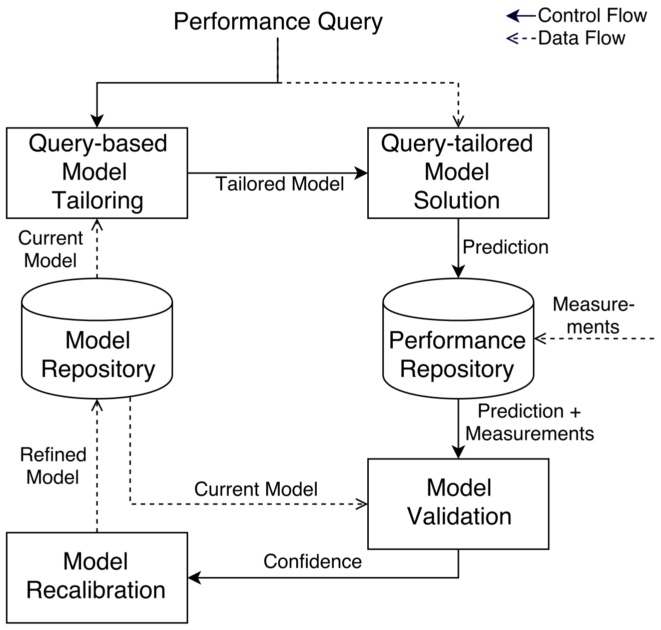

Fig. 1. Overview

In order to tailor the model to a specific query, the query-based model tailoring abstracts and/or deletes model elements, which have little to no impact on the requested metrics. This speeds up model solution and makes the analysis of large systems feasible. The tailoring process continuously adapts itself by evaluating the prediction accuracy of previously tailored models.

This tailored model is then forwarded to the model solution component. Based on the requested metrics and the time-sensitiveness of the query, the model solution component selects the most suitable of several existing solving methods for the specific query. The achieved prediction accuracy as well as the time-to-result, i.e., the time a solution approach requires to answer a performance query, of the solution approach are saved in the performance repository, creating a knowledge base to support future trade-off decisions.

In addition to the query results, the performance repository stores online monitoring data from the observed system. The repository can then be used to continuously analyze and validate the model in order to develop awareness of the model about its own accuracy. If the model detects inaccuracies, a recalibration is triggered.

The recalibration is concerned with extracting and parameterizing the model representation of the system. This recalibration takes measurement history and previous model learning attempts into account to develop system-specific knowledge about how the different available approaches perform in order to improve the ongoing parametrization and recalibration of the model.

All of the four steps develop their own knowledge base and hence realize their own LRA-M loop [1]. The architecture can be seen as event-based, however, the individual steps are continuously learning and evolving. In the following, we explain each of the four outlined steps in more detail.

III. Query-based Model Tailoring

Performance predictions have been shown to be useful for many purposes like bottleneck analysis [3], SLA evaluation [4], proactive auto-scaling [5] and configuration optimization [6]. For many use cases, only specific parts of the system are of interest. For example, in order to determine whether provisioning of additional resources resolves an identified bottleneck, only the throughput of the bottleneck resource is required. Similarly, to evaluate if the SLAs for a service will be violated, only the response time of the respective service is required.

However, in most cases the full system needs to be simulated to derive metrics for specific system parts. As the simulation time scales with the size of the simulated system, this leads to unnecessarily long simulation times, especially for large-scale systems. Therefore, multiple problem specific meta-models focusing on, e.g., network [12] or database performance [13] are currently used in parallel. These models depict one aspect of the system in great detail and treat the remainder of the system as a black box. However, this means multiple models have to be simultaneously maintained and concerns cutting across multiple problem domains are hard to analyze.

We aim to extend traditional architectural meta-models to provide multiple descriptions of different granularity for each component and dynamically select the appropriate description depending on the performance query. The selected granularity for each model element is based on their relation to the components for which metrics were requested. Once enough information about the impact of modeling granularities on the prediction accuracy is available in the performance repository, the rules for granularity selection can be updated.

The proposed approach increases the simulation speed, without sacrificing prediction accuracy. Since only the query-relevant parts of the model are simulated, large systems can be effectively analyzed.

IV. Query-Tailored Solution

Commonly, architectural performance models are solved by transformation to stochastic models, such as queueing networks [14], layered queueing networks [15], or stochastic process models [16]. In order to derive predictions for these models, existing analytical and simulation-based solvers can be used. Each combination of transformation and applicable solver represents a valid solution approach, capable of deriving performance predictions for the architectural performance model. Every solution approach has different properties in terms of prediction accuracy and time-to-result.

The selection of the appropriate solution approach is challenging since no static ranking of the solution approaches in terms of accuracy and time-to-result exists. Instead, the best-suited solver depends on the requested metrics, the structure and control flow of the architectural performance model, as well as the user preferences and constraints concerning the trade-off between prediction accuracy and time-to-result. Brosig et al. showed significant time-to-result differences between different simulation-based solution approaches [17]. In their experiments, the accuracy of solvers was shown to depend on the model structure.

Our goal is to predict the accuracy and time-to-result for each specific query and model combination and select the solution approach that best fits the user's accuracy and time-to-result preferences. The accuracy of a solution approach depends on static properties of the solver and the structure of the analyzed performance model. Therefore, we aim to combine expert knowledge about the solvers with an indicator of much information, and therefore accuracy, is lost in the transformation to the solution formalism for the specific model. In order to predict the time-to-result, we intend to utilize the data about previous queries stored in the performance repository. We plan to use regression analysis techniques, such as Support Vector Regression (SVR) or Multivariate Adaptive Regression Splines (MARS), to estimate the time-to-result for a specific query and performance model combination. This is made possible by the fact that the training data only contains data for this exact model or a variation thereof given that all previous queries analyze the same underlying real-world system.

Based on the accuracy and time-to-result predictions, the correct solution approach can be automatically selected for each problem instance. This improves the accuracy and time-to-result of the performance predictions and allows reusing the performance model in different contexts as the prediction process automatically adapts to new situations.

V. Model Validation

The model validation process analyzes and evaluates a performance model by comparing the model representation with actual monitoring data. This can be used to derive a confidence metric for the model. If the model detects inaccuracies, a recalibration is triggered. We distinguish between two reasons for degradation of model accuracy: (1) insufficient expressiveness of the model, i.e., the system is not depicted reasonably well in the model, and (2) evolving system state, i.e., the system was once sufficiently modeled, but now evolved (due to changing configuration or deployment).

Therefore, our model validation process is also two-fold as we envision two types of simultaneously running evaluations of the model accuracy:

1) *Historic prediction accuracy analysis*: Since all performance predictions, as well as run-time measurements, are stored in the performance repository, we can compare the predicted performance metrics with the actual measurements. This allows us to compute a confidence metric for the model. We distinguish between *active* and *passive* accuracy analysis. Passive analysis may only use stored results of previous performance queries. Active analysis autonomically issues performance queries together with automated performance tests, using for example BenchFlow [18]. By comparing the results with the predicted performance, additional insights about the model can be acquired.
2) *Self-reflective parameter analysis*: Performance model variables corresponding to observable measurement values can be directly evaluated in order to validate if the representation is accurate. Similarly, dependencies between different parameters can be tested by comparing observations of the dependent variables with the modeled dependency function. The resulting confidence values are then aggregated to reflect the holistic model confidence.

The advantage of the latter approach is that model inaccuracies might be detected and remedied before a performance query is actually issued by the user. Hence, we assume the historic prediction accuracy analysis to be better suited for detecting insufficient model expressiveness and the self-reflective parameter analysis for detecting evolving system states. Both validation processes result in a confidence value for the model. The lowest confidence is chosen, to represent the current confidence of the model in itself. If a certain confidence value (e.g., 95%) is undershot, a recalibration of the model is triggered.

VI. Model Recalibration

The recalibration process aims at adapting the current system model representation in order to improve the model accuracy. We group model inaccuracies into two types: (1) Inaccurate parametrization and (2) structural inaccuracies. Both types can be consequences of the reasons for model inaccuracies listed above. While structural inaccuracies are generally very hard to locate, wrongly or badly parameterized variables can be identified and marked by self-reflective parameter analysis. Therefore, we assume that in the case of an inaccurate model variable parametrization, the concrete model variable is known.

a) Inaccurate parametrization: Inaccurate parametrization occurs if elements are correctly modeled, but their variables are wrongly parameterized. For example, all calls of an element could be modeled, but the frequency or the ratio between those calls may be wrongly parameterized. We envision several remedies to address inaccurate variable parametrization.

- Improve current variable description: One way to improve the current variable description is by relearning an empirically observed description using additional more recent monitoring data. If this does not solve the problem, a different approach for relearning the variable description can be utilized: As the modeling of monitoring streams is a standard problem (e.g., in machine learning or forecasting), different approaches qualify. Our framework applies a collection of different algorithms from different domains, in order to enable switching between them.
- Choose alternative variable description: Apart from choosing another algorithm to characterize the variable, the model can also change the way a variable is described. Different possibilities include: (1) explicit user characterization, (2) empirical characterization based on monitoring data, or (3) description via a dependency on another model variable. This implies that dependencies can be identified and characterized, which might change the model structure. Additionally, this enables the model to actively ignore a user given characterization if a deviation from recent monitoring data is detected. Also, if several different dependencies are modeled, the model is able to switch to the most expressive one.

b) Structural inaccuracies: By structural inaccuracies we mean, inaccuracies on the model elements level, i.e., one or more elements are either missing or wrongly modeled. An example could be a component that is deployed on a different host or internally calling another component that is not modeled at all.

One challenge of structural inaccuracies is what we call *problem area detection*. As the model elements do not represent the correct structure of the system, it is difficult to locate the problem area of a structural inaccuracy. Therefore, as a first mechanism of dealing with structural inaccuracies, a re-extraction of the system structure with state-of-the-art model extraction tools (e.g., Walter et al. [10]) is triggered in response to changing deployment and/or system update. If a rediscovery of the structure does not solve the problem, the model can automatically add additional black-box components to account for unmodeled issues with response time, utilization or software parallelism. As an example, if the monitored response time of a component is always higher than predicted, an additional dummy-element is added to the component, delaying all requests by the monitored amount.

VII. Related Work

Meta-self-awareness or meta-cognition has been of interest in other research areas, such as artificial intelligence [19]. In the context of self-aware computing, Lewis et al. [20] present a reference architecture for self-aware systems, which also includes a component for meta-self-awareness. Giese et al. [21] present a generic architecture with the example of a self-aware and meta-self-aware smart home and discuss several implications of meta-reflective self-awareness. They furthermore also introduce generic architectures for individual meta-self-aware and meta-meta-self-aware systems [22]. Diaconescu et al. [23] discuss the term of meta-self-awareness in the context of collective systems and present an example for a collective intelligent transport system.

As for architectural languages, the modeling language ExecUtable RuntimE MegAmodels (EUREMA) [24], used to specify feedback loops, supports meta-self-awareness by layering feedback loops on top of each other. Other architectural languages supporting meta-self-awareness include Multi-Quality Auto-Tuning (MQuAT) [25] and the Descartes Modeling Language (DML) [26]. However, apart from high-level and architectural studies, we are not aware of any implemented meta-self-awareness in computing systems.

VIII. Conclusion and Roadmap

In this paper, we presented our vision of self-aware performance models. We identified and described four areas in which performance model creation, maintenance and solution can be improved using ideas from self-aware computing. This in turn enables a meta-self-aware computing system as the individual mechanisms of the self-aware system exhibit self-awareness themselves. We presented concrete ideas of four self-awareness processes applied in the tool-chain of a self-aware computing system.

As the query-based tailored model is input for the model solution process, we plan to realize the two steps sequentially. The same applies for model validation and model recalibration. However, we will work on query-based model tailoring as well as model validation in parallel, as these processes complement each other.

References

[1] S. Kounev, P. Lewis, K. L. Bellman, N. Bencomo, J. Camara, A. Diaconescu, L. Esterle, K. Geihs, H. Giese, S. Götz *et al.*, “The Notion of Self-Aware Computing,” in *Self-Aware Computing Systems*. Springer, 2017, pp. 3–16.

[2] N. Huber, F. Brosig, S. Spinner, S. Kounev, and M. Bähr, “Model-Based Self-Aware Performance and Resource Management Using the Descartes Modeling Language,” *TSE 2017*, vol. 43, no. 5, 2017.

[3] N. Sato and K. S. Trivedi, “Stochastic Modeling of Composite Web Services for Closed-form Analysis of their Performance and Reliability Bottlenecks,” in *International Conference on Service-Oriented Computing*, 2007, pp. 107–118.

[4] J. F. Pérez and G. Casale, “Assessing SLA Compliance from Palladio Component Models,” in *SYNASC 2013*, 2013, pp. 409–416.

[5] A. Bauer, N. Herbst, and S. Kounev, “Design and Evaluation of a Proactive, Application-Aware Auto-Scaler,” in *Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering (ICPE 2017)*, 2017, pp. 425–428.

[6] A. Koziolek, D. Ardagna, and R. Mirandola, “Hybrid Multi-Attribute QoS Optimization in Component Based Software Systems,” *Journal of Systems and Software*, vol. 86, no. 10, pp. 2542 – 2558, 2013.

[7] K. Krogmann, “Reconstruction of Software Component Architectures and Behaviour Models using Static and Dynamic Analysis,” Ph.D. dissertation, Karlsruhe Institute of Technology, 2010.

[8] M. Awad and D. A. Menascé, “Dynamic Derivation of Analytical Performance Models in Autonomic Computing Environments,” in *Proceedings of the 2014 Computer Measurement Group Conference*, 2014, pp. 159–168.

[9] F. Brosig, N. Huber, and S. Kounev, “Automated Extraction of Architecture-level Performance Models of Distributed Component-based Systems,” in *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering, ASE*, 2011, pp. 183–192.

[10] J. Walter, C. Stier, H. Koziolek, and S. Kounev, “An Expandable Extraction Framework for Architectural Performance Models,” in *QUDOS’17*. ACM, April 2017.

[11] P. Lewis, K. L. Bellman, C. Landauer, L. Esterle, K. Glette, A. Diaconescu, and H. Giese, “Towards a Framework for the Levels and Aspects of Self-aware Computing Systems,” in *Self-Aware Computing Systems*. Springer, 2017, pp. 51–85.

[12] P. Rygielski, M. Seliuchenko, and S. Kounev, “Modeling and Prediction of Software-Defined Networks Performance using Queueing Petri Nets,” in *SIMUTools 2016*, 2016, pp. 66–75.

[13] R. Osman and W. J. Knottenbelt, “Database System Performance Evaluation Models: A Survey,” *Performance Evaluation*, vol. 69, no. 10, pp. 471–493, 2012.

[14] H. Gomaa and D. A. Menascé, “Performance Engineering of Component-based Distributed Software Systems,” in *Performance Engineering*, 2001, pp. 40–55.

[15] H. Koziolek and R. Reussner, “A Model Transformation from the palladio component model to layered queueing networks,” in *SPEC International Performance Evaluation Workshop*, 2008, pp. 58–78.

[16] V. S. Sharma and K. S. Trivedi, “Quantifying Software Performance, Reliability and Security: An Architecture-based Approach,” *Journal of Systems and Software*, vol. 80, no. 4, pp. 493–509, 2007.

[17] F. Brosig, P. Meier, S. Becker, A. Koziolek, H. Koziolek, and S. Kounev, “Quantitative Evaluation of Model-Driven Performance Analysis and Simulation of Component-based Architectures,” *TSE 2015*, vol. 41, no. 2, pp. 157–175, 2015.

[18] V. Ferme and C. Pautasso, “Towards Holistic Continuous Software Performance Assessment,” in *Proceedings of the 8th ACM/SPEC International Conference on Performance Engineering*. New York, NY, USA: ACM, 2017, pp. 159–164.

[19] M. T. Cox, “Field Review: Metacognition in Computation: A Selected Research Review,” *Artificial intelligence*, vol. 169, no. 2, pp. 104–141, 2005.

[20] P. R. Lewis, A. Chandra, F. Faniyi, K. Glette, T. Chen, R. Bahsoon, J. Torresen, and X. Yao, “Architectural Aspects of Self-Aware and Self-Expressive Computing Systems: From Psychology to Engineering,” *Computer*, vol. 48, no. 8, pp. 62–70, Aug 2015.

[21] H. Giese, T. Vogel, A. Diaconescu, S. Götz, and S. Kounev, “Architectural Concepts for Self-aware Computing Systems,” in *Self-Aware Computing Systems*. Springer, 2017, pp. 109–147.

[22] H. Giese, T. Vogel, A. Diaconescu, S. Götz, and K. L. Bellman, “Generic Architectures for Individual Self-aware Computing Systems,” in *Self-Aware Computing Systems*. Springer, 2017, pp. 149–189.

[23] A. Diaconescu, K. L. Bellman, L. Esterle, H. Giese, S. Götz, P. Lewis, and A. Zisman, “Architectures for Collective Self-aware Computing Systems,” in *Self-Aware Computing Systems*. Springer, 2017, pp. 191–235.

[24] T. Vogel and H. Giese, “Model-Driven Engineering of Self-Adaptive Software with EUREMA,” *ACM Trans. Auton. Adapt. Syst.*, vol. 8, no. 4, pp. 18:1–18:33, Jan. 2014.

[25] S. Gotz, C. Wilke, S. Cech, and U. Aßmann, “Architecture and Mechanisms for Energy Auto Tuning,” *Proc. Sustainable ICTs and Management Systems for Green Computing*, pp. 45–73, 2012.

[26] S. Kounev, N. Huber, F. Brosig, and X. Zhu, “A Model-based Approach to Designing Self-aware IT Systems and Infrastructures,” *Computer*, vol. 49, no. 7, pp. 53–61, 2016.