Automated and Adaptable Decision Support for Software Performance Engineering

Jürgen Walter
University of Würzburg
97074 Würzburg, Germany

Andre van Hoorn
University of Stuttgart
70569 Stuttgart, Germany

Samuel Kounev
University of Würzburg
97074 Würzburg, Germany

ABSTRACT

Software performance engineering (SPE) provides a plethora of methods and tooling for measuring, modeling, and evaluating performance properties of software systems. The solution approaches come with different strengths and limitations concerning, for example, accuracy, time-to-result, or system overhead. While approaches allow for interchangeability, the choice of an appropriate approach and tooling to solve a given performance concern still relies on expert knowledge. Currently, there is no automated and extensible approach for decision support. In this paper, we present a methodology for the automated selection of performance engineering approaches tailored to user concerns. We decouple the complexity of selecting an SPE approach for a given scenario providing a decision engine and solution approach capability models. This separation allows to easily append additional solution approaches and rating criteria. We demonstrate the applicability by presenting decision engines that compare measurement- and model-based analysis approaches.

CCS CONCEPTS

• **General and reference → Evaluation**; **Experimentation**; **Performance**; • **Software and its engineering → Model-driven software engineering**;

KEYWORDS

Decision support, Software performance engineering, Model-based analysis, Measurement-based analysis

ACM Reference Format:
Jürgen Walter, Andre van Hoorn, and Samuel Kounev. 2017. Automated and Adaptable Decision Support for Software Performance Engineering. In *VALUETOOLS 2017: 11th EAI International Conference on Performance Evaluation Methodologies and Tools, December 5–7, 2017, Venice, Italy*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3150928.3150952

1 INTRODUCTION

Performance of software systems is relevant to multiple application areas because it has a major impact on key business indicators. Performance evaluation techniques include measurement, simulation, analytical solutions, model transformations to stochastic formalisms, algebraic solutions, and fluid approximations each having specific analysis goals and capabilities. Interoperability of evaluation techniques can be achieved by automated performance model extraction [10, 27], and by transformations of monitoring data [18] and performance models [9]. Integrating all aspects into a unified interface, declarative software performance engineering (SPE) approaches automate the whole process of deriving performance metrics [28].

SPE techniques can be compared based-on different criteria like analysis speed, accuracy, or system overhead. They come with strengths and limitations depending on user concerns and the system under test. While many alternative performance evaluation approaches exist, the choice of an appropriate SPE approach and tooling to solve a given performance concern still requires expert knowledge. Even desired statistics like distribution, mean, quantiles, percentiles, or maximum, impact the choice of SPE techniques [8]. For example, mean value analysis can be significantly faster than simulations [21]. For measurements, in-place aggregation of metrics may reduce the communication overhead. SPE can be used to evaluate a variety of metrics like utilization of resources or throughput and response times of services. The requested metrics impact the choice of an analysis approach, but also characteristics of the system under test or its model representation may impact the choice. For monitoring tools, there are technological restrictions on what languages and infrastructures can be monitored [29]. For model-based analysis, approaches come with different scalability in terms of time-to-result and memory overhead [9, 17, 19], supported model elements [9], and restrictions to certain model structures [8, 26]. To summarize, various methods, techniques, and tools for measuring, modeling, and evaluating performance properties of software systems have been proposed over the years, each with different strengths and limitations. The choice of an appropriate tooling to solve a given user concern tailored to the application scenario, requires expert knowledge. Given the complexity of selecting a solution approach, it is essential to introduce automated decision support. Existing approaches do not provide an extensibility concept to integrate new approaches and comparison attributes [8, 9]. The contribution of this paper is a methodology for automated decision support for performance engineering approaches that enables to automatically select an appropriate approach for a given system and user concern. We propose an architecture decoupling the complexity of the approach selection into a decision engine coupled with tool capability models. Besides automation, the benefit of our capability model concept is that it supports extensibility concerning solution approaches and comparison attributes. We show the applicability by presenting decision support based on capability models for measurement and model-based analysis. Our evaluation considers qualitative and quantitative attributes and discusses how to integrate new SPE approaches and comparison attributes.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
VALUETOOLS 2017, December 5–7, 2017, Venice, Italy
© 2017 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6346-4/17/12…$15.00
https://doi.org/10.1145/3150928.3150952

The remainder of this paper is organized as follows: Section 2 reviews related work. Based on the shortcomings of current approaches, we define requirements in Section 3. Section 4 describes our approach that is based on a decomposition into a decision engine and a capability model. Section 5 demonstrates decision support for model- and measurement-based approaches. Finally, Section 6 provides a conclusion.

Supplementary material including the meta-model, example models, and the decision engine implementation is available online at: http://descartes.tools/aasspe/.

2 RELATED WORK

This work is related to decision support for performance engineering approaches which can be subdivided into decision support for model-based and measurement-based analysis.

2.1 Decision Support for Model-based Analysis

Performance predictions can be performed based on different model descriptions like regression models, queueing networks, control-theoretical models, and descriptive models [23].

Qualitative comparisons of model-based solution approaches have been performed, e.g., in [3, 4, 8, 9, 14]. Works include the comparison of concrete toolchains [3, 9, 14] or methodologies [4, 8]. There are surveys focused on a single analysis type [4] or a single formalism [8, 9]. Balsamo et al. [3] provide a categorization of different model-based solution approaches according to their supported model, application domain, and life cycle phase. Koziolek [14] surveys model-based analysis approaches for component-based systems focusing on model expressiveness, tool support, and maturity. DeGooijer et al. [11] mix capabilities of formalisms and solution approaches and discuss the usability from an industrial perspective.

Quantitative analyses of solution approaches have been performed, e.g., in [9, 15, 17, 19]. Brosig et al. [9] perform a quantitative evaluation of model-driven performance analysis and simulation of component-based architectures. Müller et al. [17] quantify analysis time in relation to model parameters of a benchmark model. Rygielski et al. [20] compare different solution approaches for network performance evaluation. Summarizing, none of the above works discusses automating the choice of analysis approaches, except for [8, 9] providing decision trees. While there are advancements in SPE, none of the works discusses extensibility.

2.2 Decision Support for Measurement-based Analysis

Qualitative comparisons of measurement-based approaches have been performed in [13, 18, 29]. Gartner evaluates application performance management (APM) tool vendors on a yearly basis [13]. Kowall and Cappelli [13] define three dimensions of APM functionality: application topology discovery and visualization, application component deep dive, and user-defined transaction profiling. Okanović et al. [18] compare application performance monitoring tools concerning derivable information, like call parameters or error stack traces. According to Watson [29], key aspects to consider include programming language support, cloud support, software-as-a-service (SaaS) versus on-premise, pricing, and ease-of-use. Quantitative analyses focus on system overhead [12, 22, 25] and costs of ownership [2, 29]. Cost of ownership types for APM include: (i) software licenses, (ii) hardware and system software required to operate, (iii) deployment and instrumentation costs, and (iv) ongoing maintenance costs [2]. At this, pricing models of vendors impact the choice of monitoring infrastructures.

Monitoring may cause a significant impact on the overall performance of software systems. The MooBench [25] micro-benchmark has been developed to measure and quantify the overhead of monitoring frameworks. Monitoring of system metrics usually induces less performance overhead than application monitoring [25]. To reduce the performance impact of application monitoring, sampling can be used to avoid to collect and store detailed monitoring traces [12, 22].

To summarize, monitoring tools have been compared with respect to different dimensions. However, currently there is no approach offering a unified view and end-to-end automation. The work presented in this paper allows to aggregate existing knowledge in a unified view and provides automated decision support.

3 REQUIREMENTS

Based on shortcomings in the state of the art, we formulate requirements for the automated decision support and the comparison methodology:

- Decision support should be based on user concerns defining metrics, statistics, constraints, and optimization attributes.
- Decision support shall consider characteristics and limitations of the system under test.
- The decision support should allow for comparison of different kinds of SPE approaches including measurement- and model-based analysis approaches, analytical and simulation-based solvers.
- The decision support should allow for adjustments and extensions without introducing a dependency to already integrated approaches or comparison attributes.

4 METHODOLOGY

The choice of an appropriate solution approach to solve a given performance concern requires expert knowledge. The goal is to provide a framework that automates the decision based on configurable concerns supporting measurement- and model-based analysis while being extensible with respect to solution approaches and comparison metrics. Our methodology provides a solution to automatically filter applicable approaches and optimize for given concerns. Figure 1 presents the coarse-grained architecture of our decision framework. Our decision support is based on capability models of different approaches and a decision engine. Instances of the capability meta-model represent analysis approaches like measurement, simulation, or analytical solvers. They have to be registered in the registry. To propose a solution approach, the decision engine receives a performance concern and a description of the analyzed system as input.

Concern We define a `Concern` as a tuple of element, metric, statistic and optional accuracy constraints (*const*) and cost types to optimize for (*opt*):

$$cn = \{element, metric, statistic, const?, opt?\}.$$

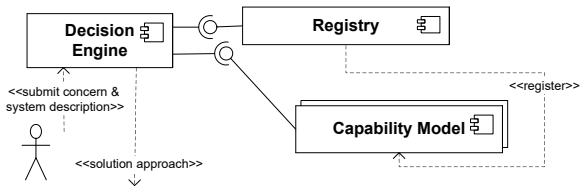


Figure 1: Decision architecture.

An exemplary concern tuple is given by [service, response time, sample, accuracy = high, time-to-result]. This tuple states that the fastest solution approach providing a sample of service response time at a high accuracy should be chosen.

System A `System` tuple describes the system under test. It may contain sets of applied languages (*language*) and middleware technologies (*middleware*), and an architectural system description (*model*). The system description is given by the tuple:

$$system = \{language*, middleware*, model?\}.$$

An exemplary system tuple is given by [(Java, Scala), (JBoss), /application/myModel.properties]. The tuple states that the application uses Java and Scala code, runs on a JBoss application server middleware, and the relative path to an architectural model of the system is given by "/application/myModel.properties".

Our system tuple offers alternative descriptions in parallel to evaluate applicability for different kinds of approaches. The system description for model-based analysis is specified by its model (denoted by its path in the file system) and meta-model, e.g., PCM [6].

The automated extraction of system descriptions including technologies required to evaluate measurement tools is more challenging. Currently, there are neither standardized representations nor interfaces to automatically derive information. While nowadays the description of applied technologies has to be put together by hand, self-describing systems might automatically provide a model of their applied technologies in the future.

Evaluating the inputs, the engine walks through all registered solution approach capability models and selects an appropriate solution strategy. At first, the decision engine decides for a given performance engineering approach if it provides the required functional capabilities to process the concern for the given system settings. In case multiple solution approaches are capable to provide the same required metrics and statistics, the decision will be made based on non-functional requirements specified in the `Concern`.

4.1 Capability Model

In the following, we present a meta-model for performance engineering tools and approaches, enabling the description of their capabilities. This enables a performance engineer to model what is provided by a solution strategy. The implementation of our capability model is based on the Eclipse Modeling Framework (EMF) [24]. This allows for the instantiation of solution strategy models by performance engineering experts using an automatically generated graphical editor [24].

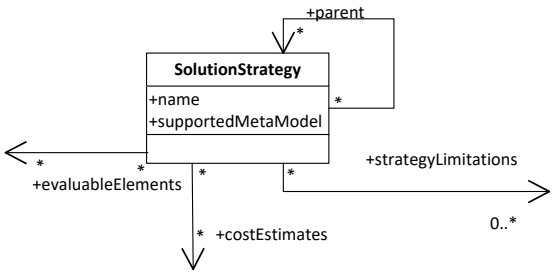


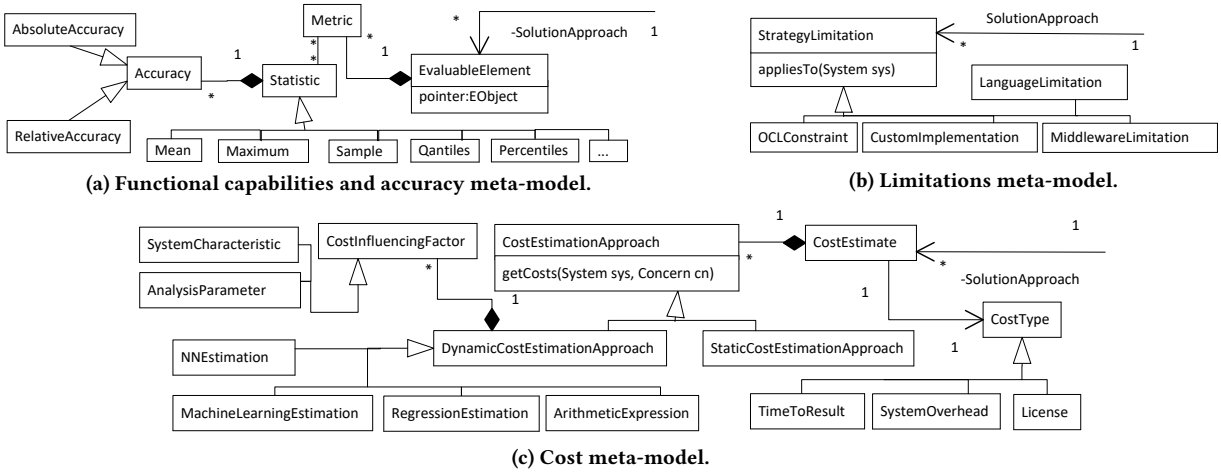
Figure 2: Solution strategy capability model root.

Figure 2 presents the root of our solution strategy capability meta-model. The root points to submodels that specify what can be derived at which accuracy, as well as the costs and limitations when applying the considered approach. The comparison of solution approaches requires the specification of a common terminology. Hence, a solution strategy links to a `supportedMetaModel` to represent the linked terminology meta-model, e.g. PCM [6].

Solution approaches can be classified into groups according to their capabilities. For example, when multiple solution approaches depend on the same model transformation, an abstract model may be employed to describe their common aspects. Another example for an analysis group with high overlap is simulation. Except for few limitations and improved time-to-result, parallel simulation provides the same capabilities like sequential simulation [26].

Our meta-model reflects similarities of models by allowing to define abstract approach capability models from which concrete models may be derived. Inheritance provides benefits if the capabilities of two solution strategies differ only in a few capabilities. In such cases, the knowledge has to be persisted only once in the form of an abstract capability model. This enables the reuse of capability specifications for multiple tools. A capability model may be defined using a hierarchical structure. Instances of an abstract capability model should include sufficient information for decision making so that a developer may specify a method's capability model just by inheritance. Abstract capability models enable the deduction and recommendation for general approaches without considering a concrete tool implementation.

4.1.1 Evaluable Elements. At first, we model what metrics and statistics can be requested. `EvaluableElements`, described in Figure 3a, define functional capabilities of a solution approach and may be extended by an accuracy description. Example types of `EvaluableElement` include service or resources like CPU or HDD. To specify the type, an `EvaluableElement` contains a pointer to define the type element of the connected meta-model terminology. An `EvaluableElement` contains a set of metrics. We modeled an initial set of metrics including, e.g., response time, throughput for services and utilization of resources. Each metric definition has to contain at least one statistic. Statistics refine supported ways to determine a metric by the given solution strategy. We integrated an initial set of common statistics, like mean, sample, maximum, quantiles, and percentiles. This set can be arbitrarily extended by implementing the `Statistic` interface. The statistic definition may be linked to a specification of the solution accuracy. The instantiations of the abstract accuracy class allow to model a maximum absolute or relative deviation from ground truth data.



(a) Functional capabilities and accuracy meta-model.

(b) Limitations meta-model.

(c) Cost meta-model.

Figure 3: Subparts of the solution strategy capability meta-model.

4.1.2 Limitation Modeling. The applicability of solution strategies can be limited by several constraints (Figure 3b). `StrategyLimitations` have to implement the `appliesTo` interface evaluating applicability based on a passed `System` description.

To evaluate limitations for model-based analysis, the passed `System` has to contain an architectural description. Then solution strategy limitations can be defined using Object Constraint Language (OCL) [24]. OCL allows to declaratively query constraints in objects. Exemplary constraints for model-based analysis are on applicable input models (e.g., for product form solutions) [8, 9] or limitations of model-transformations [9]. Besides using OCL, the evaluation of applicability can also be based on `CustomImplementation` to tailor a more efficient solution.

While concepts can be transferred, measurement tools are limited to certain technologies. To model supported languages and technologies, `LanguageLimitation` and `MiddlewareLimitation` contain a textual list of supported technologies. This allows, for example, modeling a limitation to Java technology for a monitoring tool. To evaluate if provided technologies match to a `System` description, a matching of supported tool technologies to system capabilities can be performed.

In the future, we envision a connected database of limitations storing limitations that occur frequently. This database may contain predefined `StrategyLimitations` for solution strategies in association with an architectural meta-model. This database will be linked to our capability model by a reference.

4.1.3 Cost Modeling. Solution strategies differ in several cost types like time-to-result or system overhead, as depicted in our cost meta-model in Figure 3c. Which cost type is the most relevant depends on the specific application scenario. Therefore, a `CostEstimation` has a dedicated `CostType`. For model-based analysis, this can be, e.g., `TimeToResult`. For measurement-based analysis, `SystemOverhead` and `License` costs are common comparison and optimization criteria while for model-based approaches system overhead can be considered to be zero.

Costs can either be static (e.g., fixed license costs, time-to-result always high or low) or dependent on the system and analysis configuration. Static costs can be used as a simplification when complex relationships are not known. In the more general case, the costs may depend on `CostInfluencingFactors`, which may depend either on the system characteristic or on the analysis configuration. A `SystemCharacteristic` defines a property of the system or model to be analyzed, e.g., instantiation type or count of elements, recursion depth, or a composed metric. An `AnalysisParameter` describes a parameter of the analysis setting of the given solution strategy. This can be, for example, the accuracy configuration for a simulation run, as simulation runs parameterized to be more accurate are expected to require more runtime to return the result. `CostEstimation` offers a method to predict the cost for the analysis of a given `System` and `Concern` based on the interpretation of cost influencing factors. The abstract concept of system- and concern-aware `DynamicCostEstimationApproach` can be implemented by concrete prediction approaches. This enables the integration of arithmetic expressions capturing expert knowledge and various estimation techniques, e.g., using neural networks, machine learning approaches, or regression-based approaches.

4.2 Decision Engine

Based on the capability model defined in the previous section, we can build a decision engine to decide which toolchain to use to process a given performance `Concern` for a given `System`. The decision process follows a two-step approach. First, our framework selects applicable solution approaches that come into question. Then, the applicable approaches are rated according to cost and accuracy criteria as defined in the user concern.

4.2.1 Selecting applicable approaches. Algorithm 1 decides if a given strategy is applicable for a given `Concern` and `System` description. The `isApplicableForConcern` function traverses given

capability models and checks for matches of the evaluation element, metric, and statistic defined in the concern element. This function may determine, for example, that mean value analysis is not applicable if a probability distribution is requested or that system level monitoring tools cannot provide application performance metrics. When provided in the concern definition, `isApplicableForConcern` also filters applicable approaches according to accuracy constraints. The `isApplicableForSystem` function evaluates if a solution strategy can be applied under given system settings, that is, the decision engine checks if one of the solution approach `StrategyLimitations` appears within the system, as illustrated in Algorithm 2.

Algorithm 1 Select applicable strategies.

```
1:  function getApplicable(List<SolutionStrategy> strategies,
    Concern concern, System sys)
2:      applicableStrategies ← empty list
3:      for all strategy in strategies do
4:          if isApplicableForConcern(strategy,concern) then
5:              if isApplicableForSystem(strategy,sys) then
6:                  applicableStrategies.add(strategy)
7:              end if
8:          end if
9:      end for
10:     return applicableStrategies
11: end function
```

Algorithm 2 Evaluate limitations for a system.

```
1:  function isApplicableForSystem(SolutionStrategy strategy,
    System system)
2:      for all limitation in strategy.strategyLimitations do
3:          if limitation.appliesTo(system) then
4:              return false
5:          end if
6:      end for
7:      return true
8:  end function
```

4.2.2 Rating according to concerns. After filtering applicable approaches, remaining approaches have to be rated according to their costs. In case the concern includes a cost type to optimize for, the decision engine rates applicable approaches and returns the most cost efficient approach. As users might not want to specify cost comparison attributes in every concern, we additionally allow to specify a default order of cost comparison attributes within the decision engine. This allows to omit the cost type in the concern definition. The question arises how to decide when meta-information about the system overhead, time-to-result, and accuracy is not provided for a capability model? We extend the decision logic to forward information on what is missing to the user.

5 DEMONSTRATION

In this section, we demonstrate how to apply our methodology and tooling to compare measurement- and model-based analysis approaches.

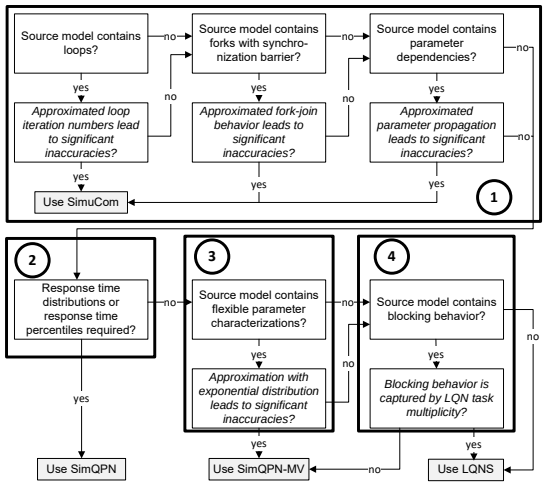


Figure 4: Solution strategy decision tree from [9]

5.1 Comparison of model-based analysis

We apply our methodology to architectural performance models as they use a terminology similar to system architecture models, compared to stochastic formalisms like Queueing Networks (QNs). We demonstrate how our approach works by comparing analysis approaches for the Palladio Component Model (PCM) formalism [6]. Supported solution strategies for PCM include SimuCOM, LQNS, SimQPN, and SimQPN MVA [9]. SimuCOM transforms a PCM instance to a process-based discrete-event simulation supporting all PCM modeling concepts [6]. A transformation to Layered Queueing Networks (LQNs) allows triggering the analytical LQNS solver [15]. A transformation to Queueing Petri Nets (QPNs) enables a simulation and a mean value analysis (MVA) using the SimQPN tool [16]. We decompose the decision tree of Brosig et al. [9] into four main steps depicted in Figure 4. In the following, we explain how to extract capability models. The results of reverse engineering the decision tree are shown in the capability models depicted in Table 1.

The first steps of the decision tree evaluate model capabilities that cannot be solved accurately by the available approaches based on transformations to stochastic formalisms. If the model contains loops, forks, or parametric dependencies that cannot be approximated, inaccuracies occur that make the approaches inapplicable. To model these limitations, constraints have to be integrated into the capability models of LQNS, SimQPN, and SimQPN MVA. In case these limitations do not apply, the tree proposes not to use SimuCOM. The decision tree implicitly captures that SimuCOM performs badly in terms of time-to-result. Our approach allows to explicitly preserve this information within in the capability models.

Then, Step 2 of the decision tree states that when the distribution of response times is required, SimQPN has to be used. Accordingly, we append the statistic type sample to the capability model for the solution strategy SimQPN. SimQPN MVA and LQNS are only capable to derive means, which can be explicitly modeled.

Step 3 models a limitation of the transformations to LQN. The containment of flexible parameter characterizations leads to significant inaccuracies. Consequently, we define a constraint in the LQNS capability model.

Analysis approach	request class: entity	metric	stat.	accuracy	costs: time-to-result	limitations model	Case
SimuCOM	serv.	resp. time	sample	high	very high	-	tree
LQNS	serv.	resp. time	mean	high	very low	no loops, no fork-join, no param. dep., no blocking beh.	tree
SimQPN	serv.	resp. time	sample	high	medium	no loops, no fork-join, no param. dependencies	tree,3
SimQPN MVA	serv.	resp. time	mean	high	low	no loops, no fork-join, no param. dep.	tree,3
SimQPN parallel	serv.	resp. time	sample	high	very low	open workload, no loops, no fork-join, no param. dep.	2,3
JMT	serv.	resp. time	mean	high	medium	no loops, no fork-join, no param. dep., ≤64 job classes	1

Table 1: Capability models for model-based analysis approaches.

Case				4	5	1,2,3,6
Analysis approach	request class: entity	metric	stat.	costs: overhead	license	limitations
DynaTrace	serv.	resp. time	sample	low	$	only Java, Node.js, .NET, MySQL, Python, Perl, Erlang, Ruby, iOS …
Instana	serv.	resp. time	sample	low	$	only .Net, Crystal, Go, Java, Node.js,PHP, Ruby, Scala
Kieker	serv.	resp. time	sample	medium	-	only Java, Perl, C#, VB6, Cobol, IEC61131-3
SPASS-meter	serv.	resp. time	sample	very low	-	sampling on high load, only Java

Table 2: Capability models for measurement-based approaches.

Further, if the source model contains blocking behavior not captured by LQN task multiplicity (step 4), LQNS cannot be used, too. So we have to define an additional constraint for LQNS that checks if the source model contains blocking behavior not captured by the LQN task multiplicity.

Using our decision engine, the capability models depicted in Table 1 allow for exactly the same selection of methods as in the decision tree in Figure 4. Moreover, our approach allows separating applicability from cost and accuracy concerns. For example, when a system model contains loops, our approach proposes SimuCOM knowing that only SimuCOM is applicable. In the case of a scenario model, that does not contain loops, forks or parameter dependencies, our approach states that SimuCOM and SimQPN are applicable and recommends SimQPN as it simulates faster. To fully automate the evaluation of limitations, we apply a simplifying generalization by restricting to *all* loops, branching actions, blocking behavior, and parametric dependencies, even though there might be analyses where respective entities have no significant effect.

Providing capabilities to depict existing decision support, the main benefit of our approach is that it provides flexibility for evolution scenarios, as demonstrated in the following use cases:

Case 1 *Adding a new solution approach.* While adding a solution approach to the tree would require to understand all approaches, our approach demands only to specify the capabilities of the new solution strategy. We added a transformation to QNs and a subsequent simulation using JMT [7]. It shares the limitations of other transformations. Moreover, the solver implementation has a technical restriction to 64 job classes which makes it impossible to simulate large scale models. However, for small models, it performs better than SimQPN in terms of time-to-result.

Case 2 *Adding a similar solver.* We reuse the transformation to QPNs and combine it with a parallel simulation using SimQPN [26]. Parallelization allows for significant speedup when the workload is specified as an open workload which is a prerequisite for efficient decomposition [26]. Capabilities for parallel and sequential simulation are very similar. This generally indicates hierarchical capability modeling which could be derived using refactoring.

Case 3 *Improvements of transformations.* As illustrated by limitations, existing model transformations often do not support all features of a source model. Extensions of transformations to support more source model features can be easily included by updating the capability models. Moreover, there are transformation alternatives based on the same source model. For example, architectural models allow to model deterministic loop counts. Transformations transform to probabilistic models for complexity reasons, but QPNs would also allow for deterministic loop counts using multiple token colors.

Case 4 *Preselection.* Our approach enables easy pre-adjustments. Some application scenarios imply constraints known in advance that should not be evaluated on every request. A preselection of approaches for certain scenarios may be realized by registering only a subset of capability models. For example, knowing that only response time distribution will be requested, we can limit the registered approaches to SimuCOM and SimQPN. Having time constraints, one might select a subset of fast solvers.

5.2 Comparison of measurement-based analysis

Measurement-based analysis approaches may use the same schema of accuracy, costs and limitations. However, their comparison focuses on different features. While time-to-result is very relevant for model-based analysis, the time required for measurements is often similar for different measurement tools. In contrast to model-based analysis, system overhead plays a critical role. System monitoring approaches, like Magpie [5] or X-Trace [1], are minimally invasive and target only network and operating system parameters. They come with the advantage of low system overhead but are not able to provide a view of internal application behavior. For APM tools, the monitoring overhead is equal to the number of transactions times the number of measurement points divided by implementation overhead in seconds — depending on the actual monitoring tool implementation. Measurement-based approaches are usually not assessed or compared according to accuracy, as there is no ground truth like for model-based analysis. Therefore, we omitted a presentation of accuracy in Table 2 — which we would consider `very high` for all approaches.

Commercial monitoring frameworks often incur significant license costs, which makes them worth to be included in the capability models of measurement-based approaches. In general, one could also model other cost types like hardware and system software required to operate, deployment and instrumentation costs, and ongoing maintenance costs. Monitoring tools are limited to the technology stacks for which they provide agents. A very important question is if a given monitoring tool can be applied to a specific system having specific technologies. This can be modeled using limitations. Table 2 compares measurement-based analysis approaches according to the discussed capabilities.

We created the capability models based on information available in the referenced literature and performed no additional measurements. The selection has been made to demonstrate several concepts without making any claims to be exhaustive. The list contains commercial APM tools for use in production systems like DynaTrace and Instana. Research prototypes support fewer languages, libraries, and platforms but are usually open source. To reduce overhead, there are sampling-based approaches [5, 12, 22]. SPASS-meter depicts an open-source implementation [12] while similar concepts run as closed source at large providers like Microsoft [5] or Google [22]. In the following, we discuss scenarios and explain how our decision engine answers or how to adjust to changes.

Case 1 *New Technology.* Consider a system or parts of it written in emerging technologies, like the new Go language. According to the modeled limitation, only Instana provides agents implemented for Go. Therefore, the decision engine proposes to use Instana.

Case 2 *Standard Technology.* The second use case considers a program written in a standard language, like Java. Java is supported by all investigated tools. Therefore, the decision engine delivers the information that all measurement tools are applicable. Ordered according to license costs, it would recommend the open source Kieker framework or SPASS-meter.

Case 3 *Debugging.* Debugging requires accurate failure detection. This translates into denial of sampling. Detailed application traces are necessary to produce a comprehensive view of the monitored system. Applicable approaches include DynaTrace, Instana, and Kieker.

Case 4 *Managing overhead.* In a high load production system, the performance monitoring overhead is critical. In such a scenario, coarse grained monitoring information on response times is sufficient. The decision support proposes an approach with low overhead, e.g., SPASS-meter.

Case 5 *Changes to pricing.* There is competition between APM vendors, which may result in changes to prices and pricing policy. The decision support can be updated by changing the cost specifications within capability models.

Case 6 *Tool updates.* The competition between vendors also results in an increasing set of supported languages and technologies. Relevant changes could be reflected by adapting limitation definitions of capability models.

5.3 Comparison of measurement and model-based analysis

Comparison of model- and measurement-based analysis can be performed extending the capability model definitions presented in Tables 1 and 2. While the comparison of most capabilities is straightforward, comparisons of accuracy and time-to-result pose challenges.

The accuracy of model-based analysis describes the deviation from the exact model solution. This does not reflect how accurate the system has been modeled. The accuracy of the model itself depends on how well model building and extraction techniques can capture the system. Relative comparisons of time-to-result for model-based approaches have been performed but there is no natural link to the time required for measurement-based analysis. In case additional measurements are required, model-based analysis usually outperforms measurements in terms of time to result. However, this depends on the experiment setup. Probably, required infrastructure costs depict a more suitable attribute to compare measurement and model-based approaches.

Besides triggering measurements and model-based analysis, additional approaches can be applied. If metrics of a system configuration have already been requested, historical logs could be queried. This provides accurate values at no system overhead costs. Based on traces of similar configurations, predictions based on transfer learning could also provide desired metrics.

6 CONCLUSION

Solution approaches and tools in SPE come with different capabilities, strengths, and limitations. Approaches allow for interchangeability, but there is lack of automated and extensible decision support. In this paper, we present a methodology for automated selection of a suitable performance engineering approach tailored a given system and user concerns. We decouple the complexity of the approach selection into a decision engine and solution approach capability models. The decision engine can filter applicable solution strategies and rank them according to the user concerns. Compared to tree-based approaches, our approach allows to easily append and modify solution strategies as the performance engineer does not rely on the knowledge of previously integrated approaches.

Also, our approach enables to easily introduce new comparison attributes like new costs, metrics or statistics.

As part of our future work, we plan to replace static cost specifications with context-aware cost predictions using machine learning. Moreover, we plan to integrate the automated decision support into our declarative framework to combine the benefits of automated selection and processing in production.

ACKNOWLEDGMENTS

This work is supported by the German Research Foundation (DFG) in the Priority Programme "DFG-SPP 1593: Design For Future—Managed Software Evolution" (HO 5721/1-1 and KO 3445/15-1). We thank Christoph Thiele for providing an initial prototype implementation of the capability meta-model.

REFERENCES

[1] 2007. *X-Trace: A Pervasive Network Tracing Framework*. USENIX Association. http://www.usenix.org/events/nsdi07/tech/fonseca.html

[2] 2010. *The Real Cost of Application Performance Management (APM) Ownership*. Technical Report. AppDynamics. https://www.appdynamics.co.uk/media/uploaded-files/1417564980/white_paper_-_the_real_cost_of_application_performance_management.pdf

[3] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. 2004. Model-Based Performance Prediction in Software Development: A Survey. *Transactions on Software Engineering (TSE)* 30, 5 (2004), 295–310. https://doi.org/10.1109/TSE.2004.9

[4] Jerry Banks, John S. Carson, Barry L. Nelson, and David M. Nicol. 2000. *Discrete-Event System Simulation (3rd Edition)* (3 ed.). Prentice Hall. http://www.amazon.com/Discrete-Event-System-Simulation-Jerry-Banks/dp/0130887021

[5] Paul Barham, Rebecca Isaacs, and Dushyanth Narayanan. 2003. Magpie: online modelling and performance-aware systems, In HotOS-IX. 85âĂŞ90. https://www.microsoft.com/en-us/research/publication/magpie-online-modelling-and-performance-aware-systems/

[6] Steffen Becker, Heiko Koziolek, and Ralf Reussner. 2009. The Palladio component model for Model-Driven Performance Prediction. *Journal of Systems and Software (JSS)* 82, 1 (2009), 3–22.

[7] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzi. 2009. JMT: performance engineering tools for system modeling. *SIGMETRICS Perform. Eval. Rev.* 36, 4 (2009), 10–15. https://doi.org/10.1145/1530873.1530877

[8] Gunter Bolch, Stefan Greiner, Hermann de Meer, and Kishor S Trivedi. 2006. *Queueing networks and Markov chains: modeling and performance evaluation with computer science applications*. John Wiley & Sons.

[9] Fabian Brosig, Philipp Meier, Steffen Becker, Anne Koziolek, Heiko Koziolek, and Samuel Kounev. 2015. Quantitative Evaluation of Model-Driven Performance Analysis and Simulation of Component-based Architectures. *Transactions on Software Engineering (TSE)* 41, 2 (2015), 157–175. https://doi.org/10.1109/TSE.2014.2362755

[10] Andreas Brunnert and Helmut Krcmar. 2017. Continuous performance evaluation and capacity planning using resource profiles for enterprise applications. *Journal of Systems and Software (JSS)* 123 (2017), 239–262. https://doi.org/10.1016/j.jss.2015.08.030

[11] Thijmen de Gooijer, Anton Jansen, Heiko Koziolek, and Anne Koziolek. 2012. An Industrial Case Study of Performance and Cost Design Space Exploration. In *Proc. of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE*. ACM, 205–216. https://doi.org/10.1145/2188286.2188319

[12] Holger Eichelberger and Klaus Schmid. 2014. Flexible resource monitoring of Java programs. *Journal of Systems and Software (JSS)* 93 (2014), 163–186. https://doi.org/10.1016/j.jss.2014.02.022

[13] Jonah Kowall and Will Capelli. 2014. *Magic quadrant for application performance monitoring*. Technical Report. Gartner.

[14] Heiko Koziolek. 2010. Performance Evaluation of Component-based Software Systems: A Survey. *Performance Evaluation* 67, 8 (2010), 634–658. https://doi.org/10.1016/j.peva.2009.07.007

[15] Heiko Koziolek and Ralf Reussner. 2008. A Model Transformation from the Palladio Component Model to Layered Queueing Networks. In *Proc. of SPEC International Performance Evaluation Workshop (SIPEW)*, Vol. 5119. Springer, 58–78.

[16] Philipp Meier, Samuel Kounev, and Heiko Koziolek. 2011. Automated Transformation of Component-based Software Architecture Models to Queueing Petri Nets. In *Proc. of International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), Singapore, July 25–27*.

[17] Christoph Müller, Piotr Rygielski, Simon Spinner, and Samuel Kounev. 2016. Enabling Fluid Analysis for Queueing Petri Nets via Model Transformation. In *Proc. of International Workshop on Practical Applications of Stochastic Modelling (PASM)*. Elsevier.

[18] Duan Okanović, Andre van Hoorn, Christoph Heger, Alexander Wert, and Stefan Siegl. 2016. Towards Performance Tooling Interoperability: An Open Format for Representing Execution Traces. In *Proc. of the 13th European Workshop on Performance Engineering (EPEW)*. Springer. http://eprints.uni-kiel.de/33526/

[19] J. F. Pérez and G. Casale. 2017. Line: Evaluating Software Applications in Unreliable Environments. *IEEE Transactions on Reliability* PP, 99 (2017), 1–17. https://doi.org/10.1109/TR.2017.2655505

[20] Piotr Rygielski, Samuel Kounev, and Phuoc Tran-Gia. 2015. Flexible Performance Prediction of Data Center Networks using Automatically Generated Simulation Models. In *EAI International Conference on Simulation Tools and Techniques (SIMUTools)*.

[21] P. J. Schweitzer, G. Serazzi, and M. Broglia. 1993. *A survey of bottleneck analysis in closed networks of queues*. Springer, 491–508. https://doi.org/10.1007/BFb0013865

[22] Benjamin H. Sigelman, Luiz AndrÃĬ Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. 2010. *Dapper, a Large-Scale Distributed Systems Tracing Infrastructure*. Technical Report. Google, Inc. https://research.google.com/archive/papers/dapper-2010-1.pdf

[23] Simon Spinner, Antonio Filieri, Samuel Kounev, Martina Maggio, and Anders Robertsson. 2017. Run-time Models for Online Performance and Resource Management in Data Centers. In *Self-Aware Computing Systems*. Springer.

[24] David Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. 2009. *EMF: Eclipse Modeling Framework 2.0* (2nd ed.). Addison-Wesley Professional.

[25] Jan Waller. 2014. *Performance Benchmarking of Application Monitoring Frameworks*. PhD Thesis. Faculty of Engineering, Kiel University. http://eprints.uni-kiel.de/26979/

[26] Jürgen Walter, Simon Spinner, and Samuel Kounev. 2015. Parallel Simulation of Queueing Petri Nets. In *Proc. of EAI International Conference on Simulation Tools and Techniques (SIMUTools)*.

[27] Jürgen Walter, Christian Stier, Heiko Koziolek, and Samuel Kounev. 2017. An Expandable Extraction Framework for Architectural Performance Models. In *Proc. of the 3rd International Workshop on Quality-Aware DevOps (QUDOS)*. ACM, 6.

[28] Jürgen Walter, Andre van Hoorn, Heiko Koziolek, Dusan Okanović, and Samuel Kounev. 2016. Asking "What?", Automating the "How?": The Vision of Declarative Performance Engineering. In *Proc. of International Conference on Performance Engineering (ICPE)*.

[29] Matt Watson. 2017. *Comparison of 18 Application Performance Management Tools*. Technical Report. Stackify.