

Towards Performance Prediction of Large Enterprise Applications Based on Systematic Measurements

Dennis Westermann*, Jens Happe*

*SAP Research, Vincenz-Priessnitz-Strasse 1, 76131 Karlsruhe, Germany

Email: {dennis.westermann|jens.happe}@sap.com

Abstract—Understanding the performance characteristics of enterprise applications, such as response time, throughput, and resource utilization, is crucial for satisfying customer expectations and minimizing costs of application hosting. Enterprise applications are usually based on a large set of existing software (e.g. middleware, legacy applications, and third party services). Furthermore, they continuously evolve due to changing market requirements and short innovation cycles. Software performance engineering in its essence is not directly applicable to such scenarios. Many approaches focus on early lifecycle phases assuming that a software system is built from scratch and all its details are known. These approaches neglect influences of already existing middleware, legacy applications, and third party services. For performance prediction, detailed information about the internal structure of the systems is necessary. However, such information may not be available or accessible due to the complexity of existing software. In this paper, we propose a combined approach of model based and measurement based performance evaluation techniques to handle the complexity of large enterprise applications. We outline open research questions that have to be answered in order to put performance engineering in industrial practice. For validation, we plan to apply our approach to different real-world scenarios that involve current SAP enterprise solutions such as SAP Business ByDesign and the SAP Business Suite.

I. INTRODUCTION

The performance (timing behavior, throughput, and resource utilization) of a software system is one of its key quality attributes. Performance is directly visible to the user and it heavily affects the total cost of ownership (TCO) for the system provider. Software Performance Engineering (SPE) [34] helps software architects to ensure high performance standards for their applications. However, applying performance engineering for large enterprise applications is a challenging task. Today's enterprise systems are usually built on a large basis of existing software (middleware, legacy applications, and third party services) and rarely developed from scratch. Furthermore, companies continuously adapt their applications to changing market requirements and technological innovations. Thus, performance analysts have to evaluate performance aspects during the whole lifecycle of the system. Although many approaches have been published in the context of software performance engineering, none of them has achieved widespread industrial use [20]. In many cases, the sheer size and complexity of a software system hinders the application of performance engineering in practice. Especially in large enterprise applications, the performance of a system is affected by a variety of parameters. Often, these parameters are

distributed across various layers (infrastructure, virtualization, database, application server, etc.) involving many different technologies. Thus, evaluating such systems is a time and resource consuming process.

Most existing approaches use established prediction models ([1], [20]) to estimate the performance of a software system. Most of them aim for predicting the performance in early lifecycle phases, especially before system implementation. This can avoid substantial costs for redesigning the software architecture. Concerning the evaluation of already existing components, the main focus of existing approaches lies on (i) the derivation or extraction of appropriate models and (ii) the estimation of resource demands / quantitative data needed to parameterize prediction models. Approaches focusing on the first issue analyze call traces [8] or apply static code analyses [22] to extract models of software systems. Approaches focusing on the second issue (e.g. [30], [21]) use benchmarking and monitoring of systems to derive model parameters. The general drawback of these approaches is that they are bound to the assumptions of the underlying prediction model [38]. For example, if a network connection is modeled with FCFS scheduling, it won't capture the effect of collisions on the network. Another important issue is scalability. The existing approaches do not scale with respect to size and complexity of today's enterprise applications. Creating performance prediction models for those systems requires considerable effort and can become too costly and error-prone as much work has to be done manually. For the same reason, many developers do not trust or understand performance models, even if such models are available. Concerning legacy systems and third party software, the required knowledge to model the systems may even not be available at all. Here, re-engineering approaches (e.g. [31], [22]) can help. However, the large and heterogeneous technology stack of such systems makes re-engineering often infeasible.

The approach presented in this paper, handles the complexity of large enterprise applications by abstracting those parts of the system that cannot be modeled or only with high effort. The goal is to capture the dependencies between the system's usage (workload and parameters) and performance (timing behavior, throughput, and resource utilization). The technical core of the approach is the Performance Cockpit, a framework for systematic performance evaluations. Around that technical core, there are four conceptual blocks: experiment definition, automated measurements, statistical inference, and model inte-

gration. The combination of the aforementioned blocks allows the performance analyst to evaluate the performance of large systems with reasonable effort.

The contribution of this paper is an approach that combines measurement based and model based performance engineering techniques to evaluate the performance of large enterprise applications. Furthermore, we outline open research question in order to put the approach into practice. We plan to validate the approach in different scenarios that involve current SAP solutions such as SAP Business ByDesign and the SAP Business Suite.

The paper is structured as follows. Section II illustrates the research challenges that arise owing to the considered systems and the chosen approach. In Section III we present the building blocks towards meeting these challenges. Section IV describes some application scenarios of the approach. In Section V, we outline related research work. Finally, Section VI concludes the paper.

II. RESEARCH QUESTIONS

In this section, we describe the main research questions addressed by our research. The general challenge is to understand the dependencies between the system's usage (workload and parameters) and performance (timing behavior, throughput, and resource utilization). The goal is to predict the performance behavior of the system in productive operation under real-world customer load. The following questions describe the steps towards accomplishing that goal using our systematic measurement approach.

A. How can we automatically identify the performance relevant parameters?

The sheer size of the considered systems bears a big challenge when answering the question how to identify the performance relevant parameters from a set of potential candidates. In large enterprise applications, a variety of potential performance relevant parameters and parameter combinations exist which span a huge search space. Thus, we have to develop an intelligent and efficient search algorithm that combines statistical analyses with the experiment setup. However, due to the huge cause of dimensionality such an algorithm might require too much time and resources. Thus, manual reduction of the search space may be necessary before starting the actual measurements. This can be accomplished by an explicit configuration of the measurements with appropriate heuristics based on expert knowledge. Another problem is that these parameters are distributed across various layers involving many different technologies. Thus, measuring the impact of the parameters can be a time and resource consuming process. Hence, there is a need for a powerful measurement framework that can be easily adapted and applied to different systems under test. Moreover, it should support/guide the performance analyst as far as possible in order to reduce the effort and the error rate.

B. How can we efficiently quantify the influence of specific parameters on software system performance?

Once we are able to identify the performance-relevant parameters of a software system, the derivation of their actual impact on the overall performance of the system is still a complex task. Especially in large enterprise applications, a variety of parameters affect the performance of a system. As a consequence, we have to find a trade-off between providing enough measurement data for the analyses and minimizing the period of measurements. Due to the sheer size of the considered systems it is not feasible to measure each possible parameter assignment. However, the number of measurements has to be large enough to provide statistically significant results and to cover all possible effects. Thus, we have to find an optimal mixture of intelligent experiment determination and efficient statistical analysis techniques to reduce the number of measurements. Another problem is that the system under test might not deliver the necessary monitoring information for all required parameters (e.g. due to the impact of monitoring on the performance of the system). Therefore, we have to infer this information from the available monitoring data.

C. How to deal with interdependencies?

One of the strengths of our goal-oriented measurement based approach is that it abstracts from system internals, meaning it abstracts the internal resources and behavior from prediction modeling. However, this black-box approach of course involves the risk that important dependencies between components and resources inside the abstracted system are not captured. For example, consider two otherwise independent web services share the same database. Here, we have to identify the interdependencies that influence performance without detailed knowledge about the system internals. Moreover, we have to find a solution that allows the explicit integration of these interdependencies in our resulting functions.

D. How to integrate our measurement based approach with existing model based approaches?

The combination of measurement based and model based performance prediction methodologies is a promising research field towards better applicability of software performance engineering in industrial practice [38]. This combination leverages from the benefits of both approaches. However, the results derived by the measurements and statistical analyses are basically mathematical functions that capture the dependency between a set of performance relevant input parameters (independent parameters) and certain performance metrics (dependent parameters). Thus, these functions cannot be directly combined with classical performance prediction approaches (such as queuing networks or stochastic process algebras [4]), or model-driven approaches (such as PCM [3] or CB-SPE [7]). As a consequence, we have to find a way to adjust the existing solutions in order to integrate our measured functions and combine the strength of both approaches.

E. How can we apply the approach using live monitoring instead of systematic benchmarks?

An important problem of performance prediction techniques concerning their applicability in industrial practice is that in most cases they are only practicable in early phases of the software lifecycle, particularly before the system goes live. However, in practice customers do not provide the necessary information about their expected workload which is an important parameter for detailed performance predictions ([19], [24]). Even if customers provide that information, it will for sure change over time which obsolesces the predictions made in early lifecycle phases. Furthermore, the system itself evolves over time or might integrate different third party services which can only be observed after system deployment. These are all effects that are hard to consider in early lifecycle performance evaluations. Thus, we have to develop a methodology that allows us to derive/adapt our performance models during productive operation of the system. In the course of this, we have to answer the questions discussed earlier in this section in consideration of the entailed restrictions in productive operation (e.g. that measurements must not affect the performance visible to the customer or the availability of the system). This causes additional challenges to the algorithms and analysis procedures for example due to the noisy data for statistical analyses or the reduced control over the system compared to a test setup.

III. APPROACH

In the following, we present our approach that aims at understanding the performance behavior of large enterprise applications in real customer environments. The main idea is to abstract from system internals by applying a combination of systematic goal-oriented measurements, statistical model inference, and model integration. Figure 1 illustrates the major building blocks of the approach.

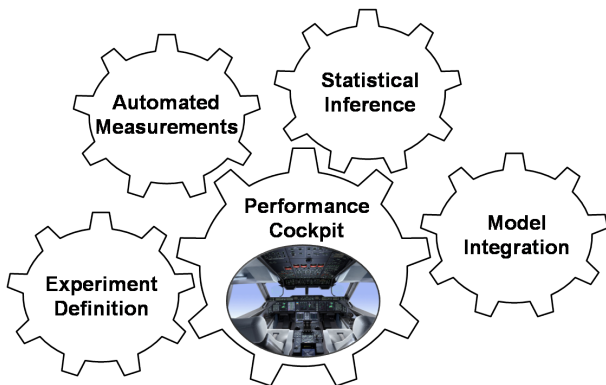


Fig. 1. Goal-oriented Systematic Measurement Approach

The technical core of the approach is our *Performance Cockpit*, a framework for systematic performance evaluations. Around that technical core, there are four conceptual blocks: *Experiment Definition*, *Automated Measurements*, *Statistical Inference*, and *Model Integration*. In what follows, we describe the building blocks of the approach in detail.

A. Performance Cockpit

Today's enterprise applications are rarely developed from scratch. On the contrary, in most cases these applications are built on a large basis of existing components such as middleware, legacy applications, or third-party services. Besides the sheer size of these systems, the resulting complexity and heterogeneity in terms of technology, distribution, and manageability complicates the application of performance evaluations. Since the performance of a system is affected by multiple factors on each layer of the system, performance analysts require detailed knowledge about the system under test. Moreover, they have to deal with a huge number of tools and techniques for benchmarking, monitoring, and data analyses. In practice, performance analysts try to handle this complexity by focusing on certain aspects, tools, or technologies within the system. However, these isolated solutions are inefficient due to the small reuse and knowledge sharing and do not provide reliable performance predictions for the overall system. The goal of the Performance Cockpit is to encapsulate knowledge about performance engineering, the system under test, and statistical analyses in a single application. Therefore, the framework implements best practices and guides the performance analyst in conducting systematic performance evaluations [15]. Moreover, the framework provides a flexible, plug-in based architecture that allows the separation of concern and supports the reuse of performance evaluation artifacts. Figure 2 illustrates the idea of the Performance Cockpit.

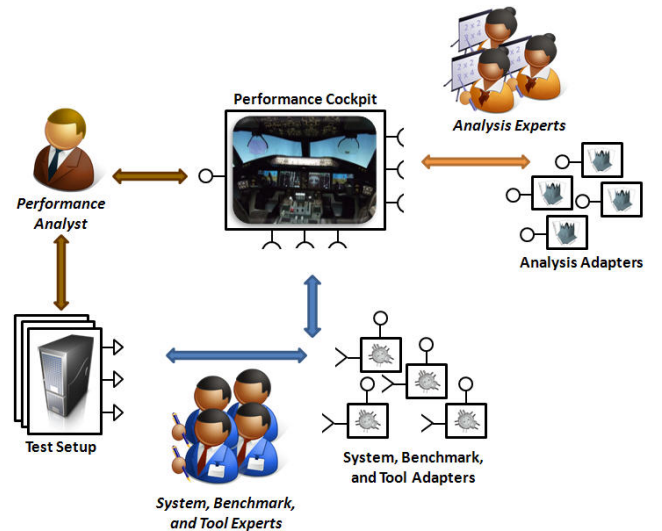


Fig. 2. The Idea of the Performance Cockpit

Each stakeholder contributes to those parts of the performance evaluation he is an expert in. The basic functionality to control the performance evaluation is provided by the framework. This plug-in based approach enables the *Performance Analyst* to reuse the adapters implemented by the *System, Benchmark, and Tool Experts* or the *Analysis Experts*, respectively. Moreover, the *Performance Analyst* can reuse adapters in multiple scenarios. Furthermore, if a component in the

system under test is changed one can easily switch the plugins without changing the actual measurement application. The resulting benefits are (i) less effort for setting up performance tests, (ii) better knowledge transfer, (iii) flexible measurement environment, (iv) better usability, and thus making performance evaluations available to a broader target group.

B. Experiment Definition

The approach introduced in this paper requires a huge number of measurements. Moreover, the approach should be applicable for various systems. In order to keep the approach feasible, we have to abstract from the concrete system under test and automate the measurements as far as possible. The Model-Driven Architecture (MDA) [27] is a design approach that allows to meet these challenges. We implement the MDA approach by designing a platform-independent meta-model for the definition of experiments. Experiment includes the system under test, workload, monitoring, analysis, measurement procedures, evaluation goals, etc. The definition of a platform-independent meta-model allows us to provide a single point of configuration to the performance analyst. Based on the meta-model, we can automatically create configurations for different parts of the performance evaluation (e.g. via model-to-model or model-to-text transformations). Figure 3 illustrates the idea.

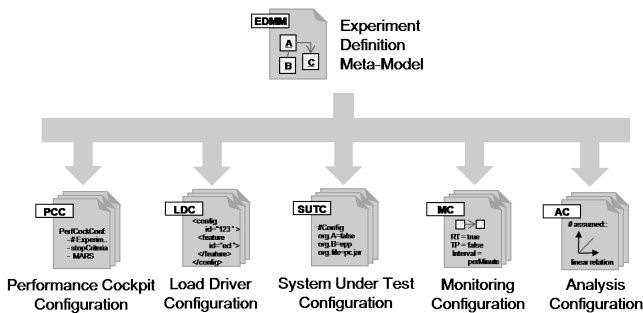


Fig. 3. ExperimentDefinitionMetaModel

The generic *Experiment Definition Meta-Model* allows us to perform multiple evaluations in a consistent and integrated way without having effect on the flexibility of the approach. Garcia, Mora, and others successfully applied such a meta-model for software artifact and process measurements [11], [28], [29]. In our approach, we focus on the configurations necessary to perform measurements of performance metrics. This includes the following points:

- *Performance Cockpit Configuration*: Information concerning the execution of measurements by the Performance Cockpit, e.g. number of experiment runs, stop criteria for the experiments, notification event receiver, and plug-in selection (load driver, system control, monitoring, analysis, etc.).
- *System Under Test Configuration*: Information concerning the setup of the system under test, e.g. system parameters, system topology including addresses, and system control information.

- *Load Driver Configuration*: Information concerning the generation of load on the system under test, e.g. the number of concurrent users, and the variation of parameters.
- *Monitoring Configuration*: Information concerning the monitoring infrastructure and behavior, e.g. monitored metrics, sampling intervals, and hold-back time of monitoring data.
- *Analysis Configuration*: Information concerning the statistical analysis of the monitored data, e.g. analysis technique, assumptions about the expected functions, expected accuracy of the results and desired output format.

C. Automated Measurements

The experiment definition meta-model described in the previous section is an approach to automate configuration and setup of measurement environments. In this section, we focus on the automated execution of measurements. Due to the size of the considered systems and the resulting huge number of necessary measurements, the automated execution is a critical success factor. In order to automate the measurements, we have to link the different areas of performance measurement by an intelligent and efficient algorithm. If setup and configuration of the system under test and the measurement environment are completed, the following steps remain for the actual measurements: determining the actual experiment setup (i.e. how to vary the parameters in each experiment run), running the experiment and measure, and analyzing. Typically, these steps are triggered manually. For example, if performance analysts want to evaluate the performance of a middleware component, they generate or adopt a certain load profile (such as provided by the SPEC benchmarks [36]) as the experiment setup and execute it, monitor the relevant metrics and parameters during execution, and finally analyze the monitored data. Often, this process is not only manually triggered but also executed only once due to the effort involved. In our approach, we will automate this process as depicted in Figure 4.

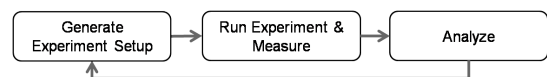


Fig. 4. Automated Measurement Process

The Performance Cockpit generates the experiment setup, automatically deploys the load drivers on the corresponding nodes, and starts the measurements. During the measurements, the Performance Cockpit captures information about the parameters and performance metrics of interest provided by existing monitoring infrastructures. The information is aggregated and saved in the cockpit's measurement data repository. The Performance Cockpit uses the data to run its statistical analyses in predefined intervals. Depending on the results of the analysis the Performance Cockpit (i) reruns the load profile analyzed in that interval (e.g. because of insufficient monitoring information) or (ii) generates and executes new load profiles (e.g. in order to detect effects not covered by the actual load profile). The presented procedure allows us to

implement highly dynamic and efficient algorithms (such as introduced by Reussner et al. [32]). This is an essential issue towards the feasibility of our approach in large, real-world enterprise applications.

D. Statistical Inference

In the previous section, we described the automated measurement process used in our approach. In the analyses phase of the process we use statistical inference [14] to capture the dependencies between the system's usage and performance. The data collected by the monitoring is used to infer (parameters of) a prediction model. In one of our recent work [13], we derived the dependencies between the usage and the performance of a message-oriented middleware using Multivariate Adaptive Regression Splines (MARS) [10] and genetic optimization [2]. While statistical inference does not require specific knowledge on the internal structure of the system under test, it might require assumptions on the kind of functional dependency between independent and dependent variables. The main difference between the multiple inference approaches is their degree of model assumptions. For example, the nearest neighbor estimator makes no assumptions on the model underlying the observations, while a linear regression makes rather strong assumptions (linearity). Most other statistical estimators lie inbetween both extremes. In general, methods with stronger assumptions need less data to provide reliable estimates, while methods with less assumptions need more data, but are also more flexible. In our approach, the concrete technique used depends on the considered problem. For example, the identification of performance relevant parameters requires other techniques than the derivation of the actual impact of a certain parameter on a certain performance metric. Additionally, the chosen technique might differ depending on the system under test, as in some cases we might have good estimators for the underlying model while in other cases the system under test is a complete black-box. Furthermore, the available monitoring information can influence the selection of an appropriate technique. In our approach, we aim for automatically selecting the appropriate method by the Performance Cockpit. However, in many cases this is not possible since expert knowledge is required. Thus, we enable the performance analyst to extensively configure the analyses using our experiment definition model (see Section III-B).

E. Model Integration

Model-driven performance prediction approaches (such as surveyed in [20]) allow to evaluate the performance of a software system prior to its implementation. The evaluation at design time has the advantage that it can avoid performance problems during implementation and testing, which can raise substantial costs for redesigning the software architecture. However, the drawback of these approaches is that they require detailed knowledge about the system under test in order to provide reliable results. The advantage of our measurement based approach is that we can apply it to nearly any system

without detailed information about its internal structure and behavior. However, the drawback of such measurement based approaches is that the system has to be available in order to conduct the measurements. This excludes the approach from answering questions like "How would the system perform if we buy another server?".

With the combination of model based and measurement based techniques, we leverage from the benefits of both approaches while overcoming the drawbacks. We target at integrating our measured functional dependencies in the Palladio Component Model (PCM). PCM is a model based performance prediction approach that targets component based, distributed systems. It is parameterizable for parameter values as well as for the deployment platform. Moreover, PCM supports the use of performance completions. Performance completions allow software architects to annotate an architecture model [12]. Model-to-model transformations refine the annotated elements by injecting low-level performance influences, e.g. of a certain middleware [18]. The completions are parametric with respect to resource demands of the annotated element. For each implementation and each execution environment the demands have to be determined explicitly. The integration of the inferred models in PCM allows us design-time performance predictions for systems that build on a large basis of existing components. In [13], we applied our approach to build a performance completion for the message-oriented middleware ActiveMQ 5.3.

IV. APPLICATION SCENARIOS & VALIDATION

The Performance Cockpit supports software architects and developers in different stages of the software lifecycle. During design time, software architects can use the Performance Cockpit to derive performance models of middleware platforms and legacy systems. In the implementation phase, developers can use the Performance Cockpit to conduct regular performance tests of their system. Furthermore, service providers can determine reliable and flexible SLAs for their services using the Performance Cockpit. In the following, we discuss possible application scenarios in more detail.

Evaluation of Design Alternatives: During design time, software architects often face architectural choices that are equivalent with respect to functionality but heavily differ with respect to performance (or extra-functional properties in general). Proper estimates of the influence of such decisions on the system's performance are essential. Design time performance predictions can avoid costly refactoring of the whole system in later development stages. However, such predictions require a detailed understanding of the middleware, third party and legacy software on which a new system is built. The Performance Cockpit enables software architects to automatically derive performance models and dependencies from systematic measurements of the systems used. Software architects can focus on the evaluation of the design decisions at hand. They can use performance completions [12] in combination with the Performance Cockpit to add low-level performance details of underlying middleware to their system under study. The

Performance Cockpit instantiates performance completions for specific implementations and configurations of a middleware platform.

In case of composable (third-party) services, software architects can use the prediction models inferred by the Performance Cockpit to assess performance characteristics of higher level services. For example, if a customer requires a special composition of business services, software architects can use the resulting model to estimate the response time and throughput of the composed business process.

Sizing and Adaptation: Sizing the underlying IT infrastructure is a critical task when deploying an enterprise application. On the one hand, the infrastructure has to provide enough resources to run the application fluently. On the other hand, purchasing and operating infrastructure resources are a substantial factor with regard to the cost-effectiveness of a company's IT landscape. Especially the trend towards providing enterprise applications as on-demand services increases the requirements on scalability and cost-effectiveness of software systems. The combination of measurement based and model based performance prediction proposed in this paper helps software architects to tailor the infrastructure to the specific needs. Thus, the approach can prevent companies from over- or undersizing their systems. In combination with appropriate cost models (such as proposed in [25]), software architects can find an efficient trade-off between costs, performance, and scalability. Moreover, the targeted performance prediction along the whole lifecycle of software systems allows early and efficient adaptations to changed workload requirements. Hence, applying the Performance Cockpit performance analysts are able to answer questions like "Can we start another instance of application X on server Y without violating existing performance agreements?".

Regression Benchmarking: Regression Benchmarking [17] (analogously to regressions testing) automatically executes a series of performance tests. The performance tests are executed on a regular basis (for instance after each nightly build). The results are summarized in a set of reports accessible to the developers. Thus, developers receive regular feedback on the performance of their system. They can directly assess the effect of changes in the implementation on software performance. The Performance Cockpit with its high degree of automation is well suited to support regression benchmarking. Its infrastructure allows the automatic execution of a series of measurements. The results of the measurements can be automatically analyzed and exported. Developers can specify the performance tests using the measurement configuration. In addition, the regularly executed benchmarks can be used to generate up to date performance models of the system under study.

Flexible SLA Specifications: In recent years, Service-level Agreements (SLAs) are gaining more and more attention. However, the specification of quality attributes in SLAs is still limited to fixed values (e.g., the response time is smaller than 2 seconds in 90% of all cases). A specification of dependencies between a service's usage and its performance

has not yet been established. In our approach, we propose a black-box specification of performance characteristics, i.e., the performance of a system is captured by a function of its usage. These black-box performance models do not contain any information about the system's internal structure. Including such models (e.g. as functions) in SLAs allows more fine-grained performance evaluations of service compositions and thus better service selections. For example, customers who require a scalable service can evaluate the available offers with respect to their expected usage profile and load. Furthermore, service providers can use the Performance Cockpit to determine reliable SLAs. The Performance Cockpit allows them to automatically execute the necessary measurements and derive the parameters for their SLAs in their environment. Based on the results, they can assess what performance they can provide. Additionally, they can use the models for internal capacity planning. For example, if a new customer requests one of their services, service providers can use the integrated prediction models to decide how the additional load will affect other customers.

For validation, we apply our approach on (i) a multi-tenant SAP ByDesign system and (ii) an SAP ERP system comprising a set of already existing SAP Enterprise Services which are composed to customer specific business processes. For each selected scenario, we will provide a validation of the prediction model (i.e. we will compare our predictions with observations). We also envision to evaluate the applicability by realizing one scenario in a larger case study. However, this depends on the feasibility and required overhead which we cannot assess in this early phase of the work.

V. RELATED WORK

This section presents current research dealing with measurement based performance analysis of software systems. Approaches that combine the different building blocks presented in this paper to an integrated, practice-oriented solution are rare. Liu et al. [26] build a queuing network model whose input values are computed based on benchmarks. The goal of the queuing network model is to derive performance metrics (e.g. response time and throughput) for J2EE applications. The approach applied by Denaro et al. [9] completely relies on measurements. The authors estimate the performance of a software system based on measurements of application specific test cases. However, both approaches simplify the behavior of an application, and thus, neglect its influence on performance. Jin et al. [16] introduce an approach called BMM that combines benchmarking, production system monitoring, and performance modeling. Their goal is to quantify the performance characteristics of real-world legacy systems under various load conditions. However, the measurements are driven by the upfront selection of a performance model (e.g. layered queuing network) which is later on built based on the measurement results. Thakkar et al. [37] describe a framework that targets the derivation of software performance

models by a series of tests. In order to reduce the required number of actually needed test runs the authors suggest to use domain knowledge or statistical analyses techniques such as Main Screen Analysis [39] and two-way ANOVA [35]. However, the authors remain open how to design such a framework and how well the suggested statistical analyses worked in their scenario. In [6], Bertolino et al. introduce an approach that verifies QoS properties of Web service implementations before their deployment. The approach is based on the automatic generation of test-beds for the Web Service under development. The authors focus on testing a single Web service while generating mock-ups for the rest of the system. In [5] they included the test-bed generation tool in a framework called PLASTIC. PLASTIC aims at enabling online and offline testing of networked applications by providing a set of tools for generating and executing tests as well as for monitoring different metrics. An approach to generate customized benchmark applications for Web service platforms is described by Zhu et al. in [42]. The approach is based on their benchmark generation tool MDABench [41].

Many measurement based approaches rely on statistical inference techniques to derive performance predictions based on measurement data. Zheng et al. [40] apply Kalman Filter estimators to track parameters that cannot be measured directly. To estimate the hidden parameters, they use the difference between measured and predicted performance as well as knowledge about the dynamics of the performance model. In [30] and [21], statistical inferencing is used for estimating service demands of parameterized performance models. Kraft et al. apply a linear regression method and the maximum likelihood technique for estimating the service demands of requests. The considered system is an ERP application of SAP Business Suite with a workload of sales and distribution operations. Pacifici et al. [30] analyze multiple kinds of web traffic using CPU utilization and throughput measurements. They formulate and solve the problem using linear regressions. Kumar et al. [23] and Sharma et al. [33] additionally take workload characteristics into account. In [23], the authors derive a mathematical function that represents service times and CPU overheads as functions of the total arriving workload. Sharma et al. [33] use statistical inferencing to identify workload categories in internet services.

VI. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed an approach for performance evaluations of large enterprise applications. Our approach focuses on the observable data and does not consider the internal structure of the underlying system. We propose a combination of model configuration, automated measurements, statistical inference and model based performance prediction in order to support performance evaluations along the whole lifecycle of a software system. The approach is realized by a flexible framework called Performance Cockpit. So far, we implemented a first prototype of the Performance Cockpit

to evaluate the performance of message-oriented middleware [13].

The approach allows software architects to create performance models for applications that include components (e.g. middleware, legacy systems, third-party services) of which they do not know or understand all performance relevant internals. Performance analysts can apply the approach to answer sizing questions, to provide guarantees in SLAs, support decisions for adaptation scenarios (e.g., moving an image from one node to another), run regression benchmarks on nightly builds, and so on. Due to the separation of concern principle and the flexible, plug-in based architecture of the Performance Cockpit, the effort to execute the aforementioned tasks is kept feasible.

In our future work, we are going to gradually answer the research questions outlined in Section II. Moreover, we will further enhance the architecture and the prototype of the Performance Cockpit framework. Currently, we are applying the approach to evaluate the performance of different types of middleware (message-oriented middleware and application servers) as well as for predicting the performance of web services and web service compositions.

Acknowledgement: This work is partially supported by the German Federal Ministry of Education and Research under promotional reference 01|G09004 (ValueGrids) and by the European Community's Seventh Framework Programme (FP7/2001-2013) under grant agreement no.216556.

REFERENCES

- [1] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, May 2004.
- [2] D. Beasley, D. R. Bull, and R. R. Martin. An overview of genetic algorithms: Part 1, fundamentals, 1993.
- [3] S. Becker, H. Koziol, and R. Reussner. The Palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82:3–22, 2009.
- [4] M. Bernardo and J. Hillston, editors. *Formal Methods for Performance Evaluation, 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, Bertinoro, Italy, May 28-June 2, 2007, Advanced Lectures*, volume 4486 of *Lecture Notes in Computer Science*. Springer, 2007.
- [5] A. Bertolino, G. Angelis, L. Frantzen, and A. Polini. The plastic framework and tools for testing service-oriented applications. pages 106–139, 2009.
- [6] A. Bertolino, G. D. Angelis, and A. Polini. Automatic generation of test-beds for pre-deployment qoS evaluation of web services. In V. Cortellessa, S. Uchitel, and D. Yankelevich, editors, *WOSP*, pages 137–140. ACM, 2007.
- [7] A. Bertolino and R. Mirandola. Cb-spe tool: Putting component-based performance engineering into practice. In *Proc. 7th International Symposium on Component-Based Software Engineering (CBSE 2004)*, pages 233–248. Springer, 2004.
- [8] F. Brosig, S. Kounev, and K. Krogmann. Automated Extraction of Palladio Component Models from Running Enterprise Java Applications. In *Proceedings of the 1st International Workshop on Run-time mOdelS for Self-managing Systems and Applications (ROSSA 2009)*. In conjunction with *Fourth International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2009)*, Pisa, Italy, October 19, 2009. ACM, New York, NY, USA, Oct. 2009.
- [9] G. Denaro, A. Polini, and W. Emmerich. Early performance testing of distributed software applications. *SIGSOFT Software Engineering Notes*, 29(1):94–103, 2004.

- [10] J. H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19(1):1–141, 1991.
- [11] F. García, M. A. Serrano, J. A. Cruz-Lemus, F. Ruiz, and M. Piattini. Managing software process measurement: A metamodel-based approach. *Inf. Sci.*, 177(12):2570–2586, 2007.
- [12] J. Happe, S. Becker, C. Rathfelder, H. Friedrich, and R. H. Reussner. Parametric Performance Completions for Model-Driven Performance Prediction. *Performance Evaluation*, In Press, Corrected Proof:–, 2009.
- [13] J. Happe, D. Westermann, K. Sachs, and L. Kapova. Statistical inference of software performance models for parametric performance completions. In *6th International Conference on the Quality of Software Architectures, QoSA 2010, Prague, Czech Republic, June 23-25, 2010, Proceedings. To Appear*.
- [14] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data mining, Inference, and Prediction*. Springer Series in Statistics. Springer, 2nd edition, 2009.
- [15] R. Jain. *The art of computer systems performance analysis*. Wiley Interscience, New York, 1991.
- [16] Y. Jin, A. Tang, J. Han, and Y. Liu. Performance evaluation and prediction for legacy information systems. In *ICSE '07: Proceedings of the 29th international conference on Software Engineering*, pages 540–549, Washington, DC, USA, 2007. IEEE Computer Society.
- [17] T. Kalibera, L. Bulej, and P. Tuma. Generic environment for full automation of benchmarking. In S. Beydeda, V. Gruhn, J. Mayer, R. Reussner, and F. Schweiggert, editors, *SOQUATECOS*, volume 58 of *LNI*, pages 125–132. GI, 2004.
- [18] L. Kapova and T. Goldschmidt. Automated feature model-based generation of refinement transformations. In *Proceedings of the 35th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pages 141–148. IEEE Computer Society, 2009.
- [19] H. Koziolok. *Parameter Dependencies for Reusable Performance Specifications of Software Components*. PhD thesis, University of Oldenburg, 2008.
- [20] H. Koziolok. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, In Press, Corrected Proof, 2009.
- [21] S. Kraft, S. Pacheco-Sanchez, G. Casale, and S. Dawson. Estimating service resource consumption from response time measurements. In *Valuetools '06: Proceedings of the 1st international conference on Performance evaluation methodologies and tools*, New York, NY, USA, 2006. ACM.
- [22] K. Krogmann, M. Kuperberg, and R. Reussner. Using Genetic Search for Reverse Engineering of Parametric Behaviour Models for Performance Prediction. *IEEE Transactions on Software Engineering*, 2010. accepted for publication, to appear.
- [23] D. Kumar, L. Zhang, and A. Tantawi. Enhanced inferencing: Estimation of a workload dependent performance model. In *Valuetools '09: Fourth International Conference on Performance Evaluation Methodologies and Tools*, 2009.
- [24] H. Li. *Workload characterization, modeling, and prediction in grid Computing*. Doctoral thesis, 2008.
- [25] H. Li, G. Casale, and T. Ellahi. Sla-driven planning and optimization of enterprise applications. In *WOSP/SIPEW '10: Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 117–128, New York, NY, USA, 2010. ACM.
- [26] Y. Liu and I. Gorton. Performance prediction of J2EE applications using messaging protocols. In G. T. Heineman, I. Crnkovic, H. W. Schmidt, J. A. Stafford, C. A. Szyperski, and K. C. Wallnau, editors, *CBSE*, volume 3489 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2005.
- [27] OMG model driven architecture, Apr. 2010. <http://www.omg.org/mda/>.
- [28] B. Mora, F. García, F. Ruiz, and M. Piattini. Model-driven software measurement framework: A case study. *Quality Software, International Conference on*, 0:239–248, 2009.
- [29] B. Mora, M. Piattini, F. Ruiz, and F. Garcia. Smml: Software measurement modeling language. In *Proceedings of the 8th Workshop on Domain-Specific Modeling (DSM'2008)*, 2008.
- [30] G. Pacifici, W. Segmuller, M. Spreitzer, and A. Tantawi. Dynamic estimation of cpu demand of web traffic. In *Valuetools '06: Proceedings of the 1st international conference on Performance evaluation methodologies and tools*, page 26, New York, NY, USA, 2006. ACM.
- [31] T. Poch and F. Plasil. Extracting behavior specification of components in legacy applications. In *CBSE*, pages 87–103, 2009.
- [32] R. Reussner, P. Sanders, L. Prechelt, and M. Mueller. SKaMPI: A detailed, accurate MPI benchmark. *Lecture Notes in Computer Science*, 1497:52–??, 1998.
- [33] A. Sharma, R. Bhagwan, M. Choudhury, L. Golubchik, R. Govindan, and G. M. Voelker. Automatic request categorization in internet services, 2008.
- [34] C. U. Smith. *Performance Engineering of Software Systems*. Addison Wesley, 1990.
- [35] M. Sopotkamol and D. A. Menascé. A method for evaluating the impact of software configuration parameters on e-commerce sites. In *WOSP*, pages 53–64. ACM, 2005.
- [36] SPEC. SPEC's benchmarks and published results. Standard Performance Evaluation Corporation, Apr. 2010.
- [37] D. Thakkar, A. E. Hassan, G. Hamann, and P. Flora. A framework for measurement based performance modeling. In *WOSP '08: Proceedings of the 7th international workshop on Software and performance*, pages 55–66, New York, NY, USA, 2008. ACM.
- [38] M. Woodside, G. Franks, and D. C. Petriu. The Future of Software Performance Engineering. In *Proceedings of ICSE 2007, Future of SE*, pages 171–187. IEEE Computer Society, Washington, DC, USA, 2007.
- [39] C. Yilmaz, A. S. Krishna, A. M. Memon, A. A. Porter, D. C. Schmidt, A. S. Gokhale, and B. Natarajan. Main effects screening: a distributed continuous quality assurance process for monitoring performance degradation in evolving software systems. In G.-C. Roman, W. G. Griswold, and B. Nuseibeh, editors, *ICSE*, pages 293–302. ACM, 2005.
- [40] T. Zheng, C. M. Woodside, and M. Litoiu. Performance model estimation and tracking using optimal filters. *IEEE Trans. Software Eng.*, 34(3):391–406, 2008.
- [41] L. Zhu, N. B. Bui, Y. Liu, and I. Gorton. Mdabench: Customized benchmark generation using mda. *Journal of Systems and Software*, 80(2):265–282, 2007.
- [42] L. Zhu, I. Gorton, Y. Liu, and N. B. Bui. Model driven benchmark generation for web services. In *SOSE '06: Proceedings of the 2006 international workshop on Service-oriented software engineering*, pages 33–39, New York, NY, USA, 2006. ACM.