Becker, S., Grunske, L., Mirandola, R., & Overhage, S. (2006). Performance prediction of component-based systems a survey from an engineering perspective.

Originally published in *Architecting systems with trustworthy components: Revised selected papers from the International Seminar on Architecting Systems with Trustworthy Components, Germany, 12–17 December 2004.*
Lecture notes in computer science (Vol. 3938, pp. 169–192). Berlin: Springer.

Available from: http://dx.doi.org/10.1007/11786160_10

Performance Prediction of Component-Based Systems

A Survey from an Engineering Perspective

Steffen Becker[1], Lars Grunske[2], Raffaela Mirandola[3], and Sven Overhage[4]

[1] Software Engineering Group, University of Oldenburg
OFFIS, Escherweg 2, 26121 Oldenburg, Germany
steffen.becker@informatik.uni-oldenburg.de

[2] School of Information Technology and Electrical Engineering, University of Queensland, Brisbane, QLD 4072, Australia
grunske@itee.uq.edu.au

[3] Dipartimento di Elettronica e Informazione
Politecnico di Milano, Italy
mirandola@elet.polimi.it

[4] Dept. of Software Engineering and Business Information Systems,
Augsburg University, Universitätsstraße 16, 86135 Augsburg, Germany
sven.overhage@wiwi.uni-augsburg.de

Abstract. Performance predictions of component assemblies and the ability of obtaining system-level performance properties from these predictions are a crucial success factor when building trustworthy component-based systems. In order to achieve this goal, a collection of methods and tools to capture and analyze the performance of software systems has been developed. These methods and tools aim at helping software engineers by providing them with the capability to understand design trade-offs, optimize their design by identifying performance inhibitors, or predict a systems performance within a specified deployment environment. In this paper, we analyze the applicability of various performance prediction methods for the development of component-based systems and contrast their inherent strengths and weaknesses in different engineering problem scenarios. In so doing, we establish a basis to select an appropriate prediction method and to provide recommendations for future research activities, which could significantly improve the performance prediction of component-based systems.

1 Introduction

In many application domains such as avionics, automotive, production-control, bio-informatics and e-business, software systems must meet strict performance goals in order to fulfill their requirements. Consequently, designers must address performance as a fundamental issue during the design and construction of software systems. This requires annotating component-based architectures with known performance qualities, and choosing fast and scalable component implementations and infrastructures. However, without an upfront effort to produce a flexible architecture during the design phase, it is rarely possible to retrofit component-based systems to significantly improve their performance.

Especially decisions taken in a late development stage, such as increasing the size of a thread pool or deploying replicated components on different hardware platforms, typically lead to a limited system performance improvement only. For that reason, unwise decisions at design-time probably render it impossible to achieve the required performance level once the system has been composed.

Current industrial practice to evaluate the performance impact of early design decisions involves the construction of prototypes, which are executed on the target deployment platform in order to measure performance properties. In this way, prototyping can help to give confidence in the resulting system performance being adequate for its needs. Prototyping is, however, expensive and time-consuming, and the results will not be valid if substantial design changes are made during implementation. Consequently, software engineering practices could be improved if software architects were able to predict the performance of the final system based on design documents without implementation details. This would reduce the effort and costs of performance prediction.

There is a substantial amount of research devoted to creating performance prediction techniques for software systems. A leading example, the software performance engineering community (SPE) [1, 2] has spent a number of years to integrate design and performance modeling activities. The developed methods are based on use case models, object and functional modeling mostly using UML-based notations. Other techniques combine analytical models with benchmarking or support model-based prototype generation. Regardless of their approach, techniques for the performance prediction of component-based systems should exhibit the following basic characteristics:

– **Accuracy.** The prediction must be accurate enough in order to provide useful results. On the other hand, a compromise between the accuracy of predictions and the analysis effort must be found in order to enable the efficient evaluation of complex applications.
– **Adaptability.** Prediction techniques should support efficient performance prediction under architecture changes where components are added/modified or replaced by different type of components.
– **Cost effectiveness.** The approach should require less effort than prototyping and subsequent measurements.
– **Compositionality.** Prediction techniques should be able to make performance predictions based on the performance characteristics of the components, which together build the system. Since component-based systems usually are structured hierarchically and consist of composite components, performance prediction techniques should be able to exploit this structure by using the analysis results on lower abstraction layers to enable the performance prediction of composite components.
– **Scalability.** Component-based systems are typically built either with a large set of simple components or utilize a few large-grain, complex components. To predict performance attributes, analysis techniques need to be scalable to handle both cases.
– **Analyzability.** Prediction techniques should not only reveal performance bottlenecks, but also give insights into possible flaws in architecture designs that are causing problems.
– **Universality.** The approach should be applicable to different component technologies with minimal modification. This enables the performance prediction of an integrated system with multiple component technologies involved.

In this paper, we analyze the applicability of existing performance prediction techniques in various software engineering problem scenarios. The aim is to highlight the strengths and weaknesses of the approaches, and reveal areas where further research is required. We start by describing a set of practical concerns which should be taken into account by performance prediction methods in section 2. Afterwards, we introduce current performance prediction methods in section 3 and classify them according to their underlying prediction technique. In section 4, we examine the individual strengths and weaknesses of the different prediction methods with respect to the before-mentioned basic characteristics and the practical concerns provided in section 2. Based on the examination of existing performance prediction techniques, we derive a variety of recommendations to improve the performance prediction of component-based systems in section 5. Finally, we conclude the paper in section 6.

2 A Taxonomy of Practical Concerns

In addition to the basic characteristics of component-based performance prediction techniques, which are mentioned in the previous section, we want to highlight some practical concerns in this section. The result will be a taxonomy of concerns, which have to be considered by performance prediction methods and their underlying prediction models. Additionally, the introduced taxonomy should also guide system architects to identify appropriate performance prediction methods with respect to their practical concerns. Both topics are investigated in detail in later sections of this paper.

It is important to stress that the problems listed in the following paragraphs are not merely scientific cases but have practical relevance. The mentioned concerns emerge in industrial scale development scenarios. This is being illustrated by the case study of an experimental Web server, which has been implemented by the Palladio group at Oldenburg University with a set of C# components. The Web server has been developed with the aim to compare and validate performance prediction methods. It supports handling basic HTTP GET and POST requests and provides an interface for the generation of dynamic HTML pages, e.g., by returning data stored in a connected database. Despite this restricted functionality and the fact that it has not that many lines of code (LOC), many concerns with today's performance prediction methods can be demonstrated by using this server. Especially, the concerns listed below emerged during the Web server development.

Third-party deployment. A major concept when looking at the performance evaluation of component-based systems is reasoning about the system properties based on the properties of the constituting components. This concept is a prerequisite to enable scalable prediction methods. Additionally, when considering Szyperski's definition [3] of a component, it is mentioned that components have to be third-party deployable. It becomes difficult to reason about extra-functional properties of components that can be deployed by third-parties, because the QoS of a component service can be only determined at run-time and only ex-post. This means that the QoS of a component service only can be determined by running the program and by measuring the QoS in question. This results from the fact that the QoS is not solely dependant on the *executable code* of the component but also on the *run-time dynamics* and the *environment* in which the component is executed in. In the web server example we have used a component accessing a database. Its QoS is significantly different if the component is deployed on the back-end server hosting the database or if it is run on a laptop with limited memory and processing power.

Third-party deployment can be regarded as super-concern, in a sense that it has an impact on all the other concerns. Thus, third-party deployment is not independent of the other concerns. Nevertheless, it is worth looking at some of the following concerns in more detail. The concerns regarded below are the usage of external services, the deployment environment, resource usage and congestion, the operational profile, and interdependencies caused by these aspects.

The third-party deployment paradigm raises an additional issue that will be elaborated later on: Approaches specifying extra-functional properties of components as constant figures will fail. This is obvious: One can only specify QoS when fixing the environment, e.g. using a reference platform, which supports the QoS figures. But then, it is only possible to deploy the component on exactly the same environment if we want to get the same QoS which is a major restriction of independent deployment. Thus, it is necessary to use specification languages and methods which are able to specify the QoS of components depending on every possible environment. During our survey, we will consequently also examine the applicability of existing approaches to QoS specification (section 3.3).

In the following, the introduced example system is investigated further to highlight the afore mentioned concerns.

External Services. The QoS of a component's service depends on the QoS of the external services called by this service. Consider for example the response-time of the component handling HTTP requests. We observe that the QoS of the `HandleRequest` method depends on the QoS of the external services the component calls, i.e., the performance of this call will never exceed the performance of the external method. For example, the attached HTML reader component (retrieving Web pages from the hard disk) needs to open a HTML file, read the data and close the file again. If opening and closing takes 200ms and processing the required data take another 100ms then the initial request will never be processed faster than 500ms. Additionally, when the request processor component is deployed in a different context, perhaps the file stream is replaced by a component delivering a network stream, it will likely perform its tasks slower due to the different QoS of the external service.

Deployment. It is not solely the constituent components of a component-based system have an impact on the component's performance, the hardware and middleware platform on which the component is deployed can make a significant difference (as already mentioned above). A component which is executed on a fast CPU will perform better than the same component which is executed on a slower CPU. Other well known influential factors are memory availability, network bandwidth, number of CPUs, performance of external storage devices (e.g. disks, optical media), and so on. Equivalent concerns apply with component-based systems that utilize software infrastructures such as middleware platforms or virtual machines. With such environments, for example, the performance of the byte code interpreter or bandwidth of the middleware server are crucial performance factors.

Given the Web server example there is a difference if its components are deployed on the .NET runtime installed on a high performing back-end server or on a desktop PC. Additionally, we have also measured that the performance of a component deployed on the Mono runtime environment (www.mono-project.com) is different from the deployment on the Microsoft implementation. This demonstrates the mentioned influences of hard- and middleware.

Resources. Software infrastructures have resource limits that constraint the performance of a component-based system. For example, the amount of threads (thread pool size), software caches/ buffers, semaphores, database connections and locking schemes must be understood in order to predict the behavior of component-based systems. Some of these factors, e.g. thread pool size, can be configured for the example Web server and their impact can be measured. Further, as a part of the deployment decisions, we could consider deploying the component redundantly on several hosts and using a load balancer to spread and monitor application load.

In general, any resource acquisition takes a certain amount of time, depending on the number of resources available. If many components have to use a rare resource it is more likely that resource conflicts will occur. In this case the component has to line up itself into a queue of components requesting the resource. Consequently, this delay due to resource conflicts need to be considered, as it influences the performance of a service call. As a result a performance prediction model has to include somehow the shared resources in environment with (virtual or physical) concurrent control flows. Additionally, the priority inversion problem could occur, where a high priority process is blocked by a low priority process, because this low priority process uses a resource which is needed by the high priority process [4].

Operational Profile. The previous section highlights additional obstacles in performance engineering. Considering the resource acquisition example, as long as the software architecture has to deal with only a single user calling its services, it is unlikely that resource conflicts will be an issue. This situation changes as soon as the component is deployed to service multiple concurrent requests. The system will have to deal with many simultaneous requests, making resource contention a major issue in terms of performance. This kind of dependency is called operational profile [5, 6].

There are additional important aspects, namely the probability of supported use cases occurring, the size and value of input parameters (which may also lead to different usage scenarios), the rate with which requests for a certain service are being made. Note the distinction between use case occurrence and request rate for a specific service. The first is from the viewpoint of the user of the system and might be translated into calls to services offered by the whole application. The second is from the viewpoint of the single components of the system. The request rate is important at every component in the system architecture even at those which cannot be called by the user directly. The translation of the usage profile in service request rates is an open field of research.

In the web server example the type of the request determines the response time. Retrieving a static HTML page stored on the hard drive is faster than retrieving a dynamic page which has to be computed from database queries. So, it is important to capture the relationship between the request types quite accurately or to model the two cases in separate use cases. Additionally the frequency and the amount of requests per unit of time

have to be taken into account, especially as many concurrent requests lead to contention of the CPU resource(s). Finally, also the size of the return type is important. Retrieving and streaming a small HTML page is faster than the retrieval of a large image.
Interdependencies. Note that there are certain interdependencies between the different aspects of a component's context. The usage of the component results in a load on the hard- and software resources as well as external services, for example the amount of concurrent threads/requests, request frequency of certain mutually exclusive resources locks, and so on. It is a major challenge to predict - utilizing the operational profile of a single service call - the operational profiles of the components which actually are involved in the handling the request. For example, a single HTTP request for a yellow page application creates several queries of the yellow page database to generate the descriptions of the matching companies for the initial query. Another example of the dependencies on external service calls can be seen in an authentification server. If this component is used on a website it mostly has to serve plain password challenges. In an enterprise context the same server might be also challenged by certificate requests.

There are two challenges arising from the preceding discussion:

- The aspects discussed above must be considered by performance evaluation techniques. Every time a certain technique abstracts one of these aspects, the prediction becomes less accurate or in certain cases totally wrong.
- If we want to support QoS prediction in the context of third-party deployable components there has to be a large specification of the respective components. There has to be a specification of the component which allows the estimation of QoS in *different* contexts. Usually this kind of specifications contain a lot of information enabling the component user to evaluate the performance and contextual aspects in a parametric way. The information can only be specified by the component producer. This is a strong assumption and appears infeasible without extensive tool support.

3 Classification of Approaches

As already outlined in the introduction, the integration of quantitative evaluation into the software development processes is an important activity to meet extra-functional, and in particular performance requirements. Balsamo et al. [7] presents a survey of different approaches for model-based performance evaluation. The proposed classification is based on the type of the performance model (Queueing Networks, Petri Nets, Process Algebras, Markov Processes), the applied evaluation method (analytical or simulative) and the presence of automated support for performance prediction. Some of these approaches have also been extended to deal with component-based systems.

In this section, we present a (short) survey of the existing approaches for predictive performance analysis of component-based systems. We distinguish between quantitative (section 3.1) and qualitative (section 3.2) analysis. Furthermore, quantitative techniques are categorized by the kind of techniques used (measurement-based, model-based, combination of measurement-based and model-based). Figure 1 visualizes the different approaches together with the cross-cutting aspect of performance specification (dealt with in section 3.3), whose existence plays a key role for a successful application of performance prediction techniques.

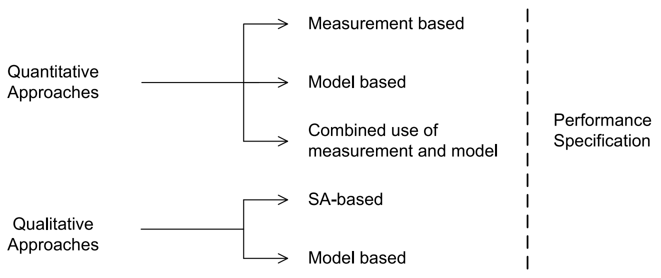


Fig. 1. *Approaches overview*

A comparison of these approaches with respect to the characteristics and concerns described in sections 1 and 2 will be presented in the next section.

3.1 Quantitative approaches

Measurement-based: Measurement-based approaches are suitable and useful, if the focus is on quantitative evaluation of performance. In [8] (**M1**) a discussion is given about how component-based system properties may influence the selection of methods and tools used to obtain and analyze performance measures. Then, a method is proposed for the measurement of performance distinguishing between application-specific metrics (e.g., execution time of various functions) and platform-specific metrics (e.g., resource utilization). The automation of the process of gathering and analyzing data for these performance metrics is also discussed. The major drawbacks of this approach are that it is only suitable for already implemented systems and the obtained results do not show general applicability of the approach.

A different approach that partially overcomes these difficulties is presented in [9] (**M2**), where starting from a specific COTS middleware infrastructure, in a first step, performance measures are collected empirically. Afterwards, in a second step, the obtained results are elaborated to extend their validity to a more general setting. The proposed approach includes a reasoning framework for understanding architectural trade-offs and the relationship between technology features and the derivation of a set of mathematical models describing the generic behavior of applications using that specific COTS technology. An inherent limitation of this approach is that it leads to sound results only for a specific hardware platform.

Denaro et al. [10] (**M3**) describes and evaluates a method for testing performance of distributed software in the early development stages. Their method takes into account the impact of the middleware used to build a distributed application. To this end the authors use architecture designs to derive application-specific performance test cases. These test cases are then executed on the available middleware platform and used to improve performance prediction in the early stages of the development process.

Model-Based: An approach, which includes predictability of performance behavior of component-based systems is presented in [11] (**MB1**). The basic idea of this approach is that the "behavior of a component-based system must be compositional in order to be scalable". To fulfill this requirement, in addition to the descriptions of the functional behavior, performance specifications are also included in component specifications. The paper outlines, how classical techniques and notations for performance analysis are either unsuitable or unnatural to capture the performance behavior of generic software components, and points out that, "performance specification problems are so basic that there are unresolved research issues to be tackled even for the simplest reusable components". A first attempt towards a compositional approach to performance analysis is then presented, mainly based on the use of formal techniques. However, as the authors argue, an engineering approach to predictability on performance is a necessary ingredient to ensure predictable components.

The papers [12–14] (**MB2**) propose a prototype enabled prediction technology, called PECT that integrates component technology with analysis models. The main goal of PECT is to enable the prediction of assembly level properties, starting from certifiable components, prior to component composition. In fact PECT is described as a "packaging of engineering methods and a supporting technical infrastructure that, together enable predictable assembly from certifiable components".

Bertolino and Mirandola introduce in [15, 16] (**MB3**) the CB-SPE framework: a compositional methodology for component-based performance engineering and its supporting tool. CB-SPE is based on the concepts and steps of the SPE technology, and uses OMG's SPT profile [17]. The technique is compositional: It is first applied by the component developer at the component layer, achieving a parametric performance evaluation of the components in isolation; then, at the application layer, the system assembler uses a step-wise procedure for predicting the performance of the assembled components on the actual platform.

In [18, 19] (**MB4**) a specific performance evaluation technique, layered queueing networks, is applied to generate performance models for component-based systems. To achieve this goal an XML-based language is defined that describes performance models of both software components and component-based systems. A model assembler tool starting from component sub-models automatically generates a layered performance models that can be solved by use of classical techniques.

In [20] (**MB5**), Balsamo and Marzolla present a simulation environment, where starting from Use Case, Activity and Deployment diagrams with RT-UML annotations (augmented, in some cases, to better fit performance features) a discrete-event C++ simulation program is derived. The transformation methodology is close to a one-to-one mapping from elements of UML model to elements of the simulator, so that the structure and the dynamics of the simulator closely follow the structure and the behavior of the UML model.

Eskenazi et al. [21] presents a method for the "Analysis and Prediction of Performance for Evolving Architectures" (APPEAR) that combines both structural and statistical techniques in a flexible way. It allows a choice of, which parts of the component are structurally described, modeled and simulated, and which parts are evaluated statistically. Additionally, the same authors present in [22] (**MB6**) a stepwise approach to predict the performance of component compositions. ”The approach considers the major factors influencing the performance of component compositions sequentially: component operations, activities, and composition of activities. During each analysis step, various models - analytical, statistical, simulation-based - can be constructed to specify the contribution of each factor to the performance of the composition. The architects can choose which model they use at each step.”

A simulation-based approach for predicting real-time behavior of an assembly based on models of its contained components is proposed by Chaudron et al. in [23] (**MB7**). The presented method deals with the main aspects of real-time systems such as: mutual exclusions, combinations of periodic and aperiodic tasks and synchronization constraints. Additionally, the simulator provides data about the dynamic resource consumption and real-time properties like response time, blocking time and number of missed deadlines per task.

In [24] (**MB8**) a compositional component performance model based on parametric contracts is presented. The approach allows for parameterization with context dependencies in order to model the performance of a single component that depends on the performance properties of the environment by using so-called service effect automata. These automata describe the call sequence of the external services on which a component service depends.

Combined use of measurement and model-based approaches: Menasce et al. [25] (**MBM1**) proposes a QoS-based approach to distributed software system composition and reconfiguration. This method uses resource reservation mechanisms at the component level to guarantee soft (i.e., average values) QoS requirements at the software system level. Different metrics can be used for measuring and providing a given QoS property, such as response time, throughput, and concurrency level. Specifically, the method relies on the definition of QoS-aware components, where a client component can request a service with a certain QoS level. In case the server is able to provide this QoS level, it commits itself to do so; otherwise a negotiation is started until an agreement on a new QoS level is reached. The method implementation is based on the combination of queueing models and measurement techniques.

In [26] (**MBM2**) a methodology is presented, which aims for predicting the performance of component-oriented distributed systems both during development and after the system have been built. The methodology combines monitoring, modelling and performance prediction. Specifically, performance prediction models based on UML models are created dynamically with non-intrusive methods. The application performance is then predicted by generating workloads and simulating the performance models.

In [27] (**MBM3**) an approach to predict the performance of component-based applications during the design phase is presented. The proposed methodology derives a quantitative performance model for a given application using aspects from the underlying component platform, and from a design description of the application. The results obtained for an EJB application are validated with measurements of different implementations. Using this methodology, it is possible for the software architect to make early decisions between alternative application architectures in terms of their performance and scalability.

3.2 Qualitative approaches

In this section we shortly describe some approaches which evaluate the quality of component-based systems either based on the affinity between software architecture and software components or exploiting the principles of the model driven engineering. The common characteristic of these approaches is to consider qualitative analyzes that are derived from an attribute-based style or trough ”screening questions” and are meant to be coarse-grained versions of the quantitative analysis that can be performed when a precise analytic model of a quality attribute is built.

SA-Based: A qualitative approach to performance analysis of component-based systems is undertaken in [28], where the affinity between Software Architecture (SA) and Software Component (SC) technology is outlined and exploited. This affinity is related to different aspects: (i) the central role of components and connectors as abstraction entities, (ii) the correlation of architectural style and component model and frameworks, (iii) the complementary agendas followed by the SA and SC technologies: enabling reasoning about quality attributed, and simplifying component integration. Therefore, the basic idea of these approaches is to develop a reference model that relates the key abstractions of SA and component-based technology, and then to adapt and apply some existing SA analysis methods, such as SAAM, ATAM and QADP.

Model-based: The basic idea of model-driven engineering (MDE) is to create a set of models that help the designers to understand and to evaluate both the system requirements and its implementation. A key point for a successful application of an MDE-based process is the integration of orthogonal models taking into account cross-cutting aspects such as the application’s performance. The following approaches are mainly descriptive and focus on paths leading to the construction of different performance models. A crucial issue for the application of MDE techniques is the existence of automatic tools allowing model transformations from design models to analysis-oriented models.

Solberg et al. [29] outlines the need to incorporate QoS specification and evaluation within a MDA-based approach at a more abstract level and at the platform-specific level. In this view, the model transformations, the code generation, the configuration and deployment should be QoS-aware. Ideally the target execution platform should be also QoS-aware.

Grassi and Mirandola [30] present an approach for the predictive analysis of extra-functional properties of component-based software systems. According to a model-driven perspective, the construction of a model that supports some specific analysis methodology is seen as the result of a sequence of refinement steps, where earlier steps can be generally shared among different analysis methodologies. The focus is mainly on a path leading to the construction of a stochastic model for the compositional performance analysis, but some relationships with different refinement paths are also outlined.

To facilitate extra-functional analysis in the design phase, automatic prediction tools should be devised, to predict some overall quality attributes of the application without requiring extensive knowledge of analysis methodologies to the application designer. To achieve this goal, a key idea is to define a model transformation system that takes as input some ”design-oriented” model of the component assembly and (almost) automatically produces as a result an ”analysis-oriented” model that lends itself to the application of some analysis methodology. However, to actually devise such a transformation, one must face both the heterogeneous design level notations for component-based systems, and the variety of extra-functional attributes.

In this perspective, the work in [31, 32] describes an intermediate model called Core Scenario Model (CSM), which can be extracted from an annotated design model. Additionally a tool architecture called PUMA is described, which provides a unified interface between different kinds of design information and different kinds of performance models, for example Markov-models, stochastic Petri nets and process algebras, queues and layered queues. Petriu et al. [33] proposes a transformation method of an annotated UML model into a performance model defined at a higher level of abstraction based on graph transformation concepts, whereas the implementation of the transformation rules and algorithm uses lower-level XML trees manipulations techniques, such as XML algebra. The target performance model used as an example in this paper is the Layered Queueing Network (LQN).

A different approach is described in [34]. This approach defines a kernel language with the aim to capture the relevant information for the analysis of extra-functional attributes (performance and reliability) of component-based systems. Using this kernel language a bridge between design-oriented and analysis-oriented notations could be established, which enables a variety of direct transformations from the former to the latter. The proposed kernel language is defined within a MOF (Meta-Object Facility) framework, to allow the exploitation of MOF-based model transformation facilities.

3.3 Performance Specification

A key point for a successful application of quantitative validation of performance properties during component-based software development is the existence of languages allowing performance specification when designing a component-based system both at component and at assembly level.

A UML Profile for Schedulability, Performance and Time (SPT Profile) has been proposed and adopted as an OMG standard [17] as a response to the exigencies of introducing in UML diagrams quantifiable notions of time and resources usage. The SPT Profile is not an extension to the UML meta model, but a set of domain profiles for UML. Basically, the underlying idea is to import annotations in the UML models, which describe the characteristics relative to the target domain viewpoint (performance, real-time, schedulability, concurrency). In this a way various (existing and future) analysis techniques can usefully exploit the provided features. In fact, the SPT profile is intended to provide a single unifying framework encompassing the existing analysis methods, still leaving enough flexibility for different specializations.

Zschaler in [35] investigates the possibility to define a framework, which can be used to provide semantics for extra-functional specifications of component-based systems, by explaining also how the different parts of a component-based system cooperate to deliver a certain service with certain extra-functional properties. The claimed objectives are ”To allow application developers to use Component-Based Software Engineering to structure their applications and thus lower the complexity of the software development process while at the same time enabling them to make use of proven and tested theories for providing extra-functional properties of those applications.”

In [36], the authors define a simple language, based on an abstract component model, to describe a component assembly, outlining which information should be included to support compositional performance analysis. Moreover, a mapping of the constructs of the proposed language to elements of the UML Performance Profile is outlined, to give them a precisely defined ”performance semantics”, and to get a starting point for the exploitation of proposed UML-based methodologies and algorithms for performance analysis.

In [37] the QoS modeling language (QML) is described. The QML is used to specify QoS attributes for interfaces, operations, operation parameters, and operation results. It is based on the fundamental concepts of contract types, contracts and profiles. Contract types are utilized to specify the metrics used to determine a specific QoS concept. Contracts are used afterwards to specify a certain level of the metrics of a contract type. The linking between contracts and interface methods, operation parameters or results is done via QML profiles. There is a conformance relation defined on profiles, contracts, and constraints. The conformance is needed at runtime, so that client-server connections do not have to be based on an exact match of QoS requirements with QoS properties. For example, if the client requests a response time of less than 5ms the server has to provide exactly a response time of less than 5ms. Nevertheless, instead of exact matches, a service is allowed to provide more than what is required by a client. In the example, the server is also allowed to provide a response time of less than 2ms as this is conforming to the required 5ms requested by the client.

4 Abstract Comparison

Table 1 relates the different performance prediction approaches described in section 3 with (a) the concerns to component-based systems described in section 2 (columns C in the table) and (b) with general characteristics of performance (prediction) techniques described in section 1 (columns A in the table). The comparison has been carried out only for quantitative approaches since qualitative methods pose themselves on the different perspective to give only qualitative insights about the performance of component-based systems.

Each row in the table refers to a specific methodology. The considered methods are grouped according to the categories introduced in section 3 (first column) and each methodology is identified by the assigned labels (second column), by the author’s names (third column) and by the reference paper(s)(fourth column).

To quantify the fulfillment of the concerns C (columns 5-9) and the characteristics A (columns 10-16) we have adopted a coarse-grained classification, i.e.,: High, Medium, Low, Absent, since the considered methods are often described with a low level of detail. Moreover, in some cases, the description is carried out at an high abstraction level that is not sufficient to quantify the relationships for the factors A and C. In these cases we have inserted in the table an educated guess followed by a question mark. Additionally, dealing with aspect A7, we have considered the existence of an automated tool or framework for the derivation of performance characteristics as a good starting point.

In the remaining part of this section we describe in some details each row of table 1 presenting how each methodology deals with factors A and C.

Let us consider, for example, the measurement based approaches (M1-M3): M1 is a introductive study and is simply based on the monitoring of single applications, while M2 and M3 seem more promising to deal with component-based systems. Both of them take middleware details into account and consider J2EE application with EJB containers. To deal with the concerns factors C4 and C5, M2 defines application-specific behavioral characteristics through the design of a set of test-cases. The performance of the server side is characterized using these tests with parameter settings concerning the transaction type and frequency, the database connection, and the pool size. Instead M3 tries to model usage scenarios relevant to performance through the modeling of workload in terms of number of user and frequency of inputs. The interactions among distributed components and resources are studied according to whether they take place between middleware and components, among components themselves or to access persistent data in a database.

Considering the general foreseeable characteristics of prediction techniques, the methods based on measurements show good value for the accuracy aspect, while they exhibit quite low values for all the other features. Actually, since they require already implemented systems their cost effectiveness is low. Moreover, they are often platform-specific and this fact limits the adaptability, the scalability and the analyzability. The universality facet in these approaches is completely absent, because the implementation and the analysis are completely manual and no automatic support is provided.

The weak point of model-based approaches is the lack of empirical studies for the validation of the predicted results. As a consequence the column A1 regarding the accuracy of the obtained results is filled-in only with educated guesses.

MB1 is one of the first papers addressing the importance of a performance engineering approach for component-based systems, and this paper specifically outlines issues like compositionality, performance specification and usage profile definition (as shown by the quite good values in the table). However, the proposed formal approach is not supported by an automated framework and seems to be only a good reasoning tool rather than a method to be applied in an industrial software development context.

MB2's aim at integrating component technology with analysis models, allowing analysis and prediction of assembly-level properties prior to component composition. It is based on the definition of an analytic interface, that takes into account the component technology, but spans aspects related to the use of some analysis methodology to support predictions about quantitative properties of the system. This approach is very attractive and shows high values in the table. It is supported by an automatic framework as well as it includes also a first ”measurement and validation environment” to validate the analysis results and to give informed feedback to the design team. A weak point is that the performance model does take usage profiles and resource contention aspects into account.

MB3, MB4 and MB5 are based on the use of UML models as design notation augmented with performance annotations compliant to the SPT profile [17]. These approaches are all supported by (semi-)automatic frameworks and provide good (also very good) values for general properties of prediction techniques. They differ in the

Table 1. Performance prediction of Component-Based Systems - comparison of quantitative approaches

	Method	Authors	Reference	Third party deployment C1	External services C2	Deployment and resources C3	Usage Profiles C4	Interdependencies C5	Accuracy A1	Adaptability A2	Cost-effective A3	Compositionality A4	Scalability A5	Analyzability A6	Universality A7
Measurement	**M1**	Yacoub	[8]	M	L	L	M	L	M	L	L	L	L	A	A
	M2	Chen et al.	[9]	M	L	H	M	M-L	M	L	M	M	L	L	A
	M3	Denaro et al.	[10]	M	L	H	M	M	M	L	L	L	L	L	A
Model-based	**MB1**	Sitaraman et al.	[11]	M	L	L	M	M	L	M	L	M	L	M	A
	MB2	Hissam et al.	[12, 14]	H	H	M	M	H	M-H	H	H	H	M	H	M
	MB3	Bertolino et al.	[15, 16]	H	M	M	M	M	M?	H	H	H	M-L	H	M
	MB4	Wu et al.	[18, 19]	M	H	M	L	M	M?	H	H	M	M	H	M
	MB5	Balsamo et al.	[20]	M-L	L	L	M-L	L	M-L?	M	H	M	M	H	M
	MB6	Eskenazi et al.	[22]	M	M	M	L	L	M?	M?	M?	M	H	H	L
	MB7	Bondarev et al.	[23]	M	M	H	L	M	L?	M?	M	H	L	M	L
	MB8	Reussner et al.	[24]	M	H	L	M-L	A	M?	L	M	H	L	H	M
Model and Measurement	**MBM1**	Menasce et al.	[25]	M	M	H	H	L	H	M	M	M	L	H	M
	MBM2	Diaconescu et al.	[26]	H	M-H?	H	H	M?	H?	L	L	M	L-M?	H	M
	MBM3	Liu et al.	[27]	H	M	H	H	M	H	L	M	M	M	H	A

Legend:
H= High, M= Medium, L= Low, A= Absent, the ? denotes an educated guess

type of performance models (queueing network, layered queueing network and simulation, respectively) and in the way of addressing the concerns that arise when developing component-based systems. MB3 and MB4 consider (even if in simple way) most aspects related to component-based models, while MB5 has been conceived for a traditional software development process and scores therefore not so well for component-based systems.

In MB6 component composition is considered in terms of concurrent activities that invoke a number of component operations. At first, a detailed analysis of performance models for component operations is carried out; then an activity model is constructed through a flow graph and finally a model of the concurrent activities is obtained. The first step is based on the combination of different methods and deals well with the concerns that occur with component-based development. The second step uses traditional techniques related to flow graphs, while the third step is not detailed enough, which making the overall understanding of the characteristics of this prediction technique difficult.

MB7 assumes as starting point Robocop component models that include resource, function, behavior and executable models. The following step combines the behavior and resource consumption models with an application model constructed for possible critical execution scenarios. This combination is performed taking into account the application static structure and its internal and external events. This model serves as an input to a simulation tool, which outputs the execution behavior of the assembly with its timing properties (latency and resource utilization). Therefore, MB7 shows quite good scores for component-based development concerns, but it is specifically targeted for the Robocop component model and this fact decreases its adaptability and scalability values.

MB8 uses components described in the Palladio component model, which includes service effect specifications (gray box component view) with service effect automata. Additionally, annotations of the probability density distribution of the response times of the external service calls and the internal calculations are used. Consequently, it gets good marks for the inclusion of external dependencies and compositional reasoning. Nevertheless, in so doing it regards hardware dependencies only implicitly. Also the usage profile is used solely to estimate probabilities of certain control flows. Multiple threads or concurrent resource utilization is disregarded.

The methods based on the combination of measurement and modeling seems to have the capabilities to combine the different concerns Cs and As in Table 1, however a weak point is represented by a low level of adaptability and scalability due to the specifics of the selected platform.

MBM1 focuses on component specifications rather than on the analysis, it consists of the definition of component QoS-aware capable of engaging QoS negotiation with other components in a distributed environment. This approach seems to be very promising, because it allows to cope with external services, deployment and usage profile in a good way. The accuracy of the obtained results are validated in an experimental environment and the presence of an automated tool for performance model generation and analysis increases the value/ attractiveness of this approach. The major drawback is the low scalability, due to the simplicity of the treated models. The models are hardly able to cope with complex systems.

MBM2 presents a framework composed of a module, which collects run-time performance information on the software components and on the software application execution environment. This module is implemented for J2EE applications and EJB container and allows also the execution of performance analyses. This is achieved with an instrumentation of the software components by a proxy layer that lowers the cost-effectiveness of the approach. Additionally, there is also a partially implemented adaptation module, which aims to solve performance problems through the selection of different, functionally-equivalent, components. The two modules are supposed to operate in an automated feedback-loop. Currently, this framework is only partially implemented and automated. This is reflected by some low (or guessed) values in the table. However, it is the first automated framework including both component and application layer, which contains an optimization module. Consequently this approach get high values for both aspects: As and Cs.

MBM3 is based on the modeling of an application in terms of component interactions and demands placed on the component container; its parameterization is carried out based on the definition of a performance profile of the container and the underlying platform obtained through benchmarking. In this way the factors Cs can be fulfilled in a quite satisfactory way. The accuracy of the obtained results is validated in a real setting and thus theoretically justified. In addition, the other A factors show medium to high values, except for the adaptability, because the method is only presented and validated in an EJB context. Furthermore, the universality of the approach can be questioned, since the method has to be applied manually and thus the results are based on the expertise and skills of the design team.

To summarize, figure 2 depicts the different scores of the discussed techniques with respect to concerns As and Cs. Roughly speaking, we can observe how measurement-based methods show low values for characteristics A while deal quite well with concerns Cs. On the contrary, the model-based approaches deal quite well with A factors while show Low or Medium values handling factors C. The joint use of models and measurement techniques combine the potentialities of both methods. A detailed comparison of different techniques would require performing some common validation experiment with the various tools and methodologies. Our feeling is that working on simple examples could lead to misleading results because of lacking of critical aspects inherent to component-based systems. To overcome this problem we are working towards the definition of a reference system including the main characteristics and concerns of a typical component-based applications and our long-term goal is to use this system to compare and validate the various performance approaches.

Although several approaches have been proposed and applied we are still far from seeing performance analysis as an integrated activity into component-based software development, both for the novelty of the topic and for the inherent difficulty of the problem.

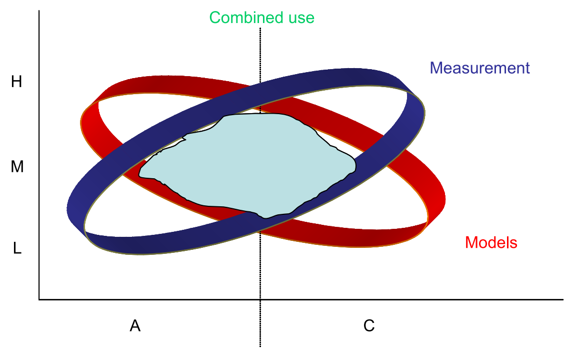


Fig. 2. *Models and measurement techniques*

5 Recommendations

In the last sections, we have stressed the importance of performance predictability in component-based development approaches as ”the ability to reason about application behavior from the quality attributes of the components and the components’ interconnections” [38]. In the following we summarize the lessons learned from the review of the current performance predictions techniques and we offer some suggestions for improvement of the performance predictability of component-based systems.

Component-based development is a new paradigm for the development of large complex software-intensive systems, but while the functional properties of the systems have been extensively dealt with both from industrial and academic communities, the quality (performance) aspect of the software products are not adequately addressed [7]. As described in section 3, in the last few years some attention has been paid also to include quantitative performance validation within component-based software development processes. The existing approaches are based on models, which has proven good predictive qualities, but they are often not considering the characteristics of component-based systems; on the other hand, measurement-based approaches are able to tackle the component-based development concerns but are frequently related to a single environment and lack of generality. Table 1 summarizes how different methods deal with the different aspect of component-based systems and could be used as a first guide in the selection of an appropriate methodology for the objective of the study/ project.

However, since the ability to predict the performance characteristics of the assembled system should be applied as early as possible to reduce the costs of late problem fixing [2], it becomes crucial to determine from the design phase whether the component-based product will satisfy its requirements. This goal can be obtained only via a rigorous design discipline and by accepting standard modelling notations as well as strict documentation and design rules, so that independently constructed components can be effectively connected and properly interact. This basic notion is central to the Design-by-Contract discipline [39], originally conceived for object-oriented systems, but even better suited for component-based development [3].

To obtain performance predictability of a component-based system several factors must be available. We devise three main features as crucial for performance prediction of component-based systems:

1 Necessity of component technology providing the means for specification of component performance taking into account its dependency both on the environment (middleware and hardware platform) and on different software resources;
2 Component selection based on the exposed performance characteristics;
3 Combination of measurement and modeling techniques embedded in an automated framework

Considering issue 1 we devise parametric performance contracts as a good way to model the dependency of a component on the ”external world” and in the following we describe the form of a parametric contract, we conceive for each service offered by a given component. Let us suppose that a given component C_i offers h $\geqslant 1$ services S_j (j=1, . . ., h). Each offered service can be carried out either locally (i.e., exploiting only the resources of the component under exam) or externally (i.e., exploiting also the resources of other components). A service S_j is defined with a set of parameters/attributes (a_1, . . ., a_m) that define/are related to its resource usage, i.e., S_j (a_1, . . ., a_m); among these attributes we can distinguish:

– constant (or non parametric attributes) such as, for example, the kind of resources required
– ”stand-alone” parametric attributes that depend only on the kind of metric we are interested in for the performance measurement (e.g., a number, if we are considering ”average” metric, a range of numbers if we are interested in best-worst case analysis);
– ”external” parametric attributes that depend also on other services.

We consider a user management component with `Login` and `Logout` methods to give examples for the above mentioned categories. The component is implemented for the J2EE platform running on a Java virtual machine. Hence, `J2EE` and `JavaVM` can be seen as constant parameters. The performance influence of the J2EE middleware platform for example was investigated in [**?**]. Information on the duration of the execution of the code of each of the user management services is part of the stand-alone parameters, e.g., the average execution time of the methods or a distribution of their response times. Additionally, the `Login` method calls an external database to verify any given username and password combination. As this call's response time adds to the `Login` method's response time the call has to be considered as external parametric attribute. Further examples for the inclusion of external service calls into a performance prediction model can be found in [**?**].

In a performance prediction model capable of modeling situations as in the example above, each component's service should be accompanied by a performance parametric contract $PerfC_i(S_j(\ldots))$ whose form depends on the kind of service parameters.

For example, if S_j is a simple service with constant attributes (a_1, . . ., a_i) and stand-alone parametric attributes ($a_i + 1$, . . ., a_m) then it can be characterized as

$$PerfC_iS_j\ (a_1, \ldots, a_m) = f_{S_j}(a_i + 1, \ldots, a_m)$$

where f_{S_j} denotes some kind of internal dependency. Otherwise, if S_j is a composite service with constant attributes ($a_1, \ldots, a_i$), stand-alone parametric attributes ($a_i + 1, \ldots, a_j$) and external parametric attributes ($a_j + 1, \ldots, a_m$) then it can be characterized as:

$$PerfC_iS_j\ (a_1, \ldots, a_m) = f_{S_j}(a_i + 1, \ldots, a_j) \oplus g_{S_j}(a_j + 1, \ldots, a_m)$$

where $\oplus$ represents some kind of composition operator.

Considering issue 2, it is obvious that to fulfill performance requirements, the system assembler will choose those components among multiple component implementations providing the same functional behavior that best fit the performance requirements.

Let us describe how the above introduced characterization can help the system assembler in the service pre-selection step where one chooses, among components offering the same service, those that provide the best performance. In fact, one can instantiate the generic $PerfC_iS_j\ (a_1, \ldots, a_m)$, with the characteristics of the adopted (or hypothesized) environment, so obtaining a set of values among which the best ones can be selected. Obviously, this kind of selection does not consider the impact of contention with other services for the use of a resource. To this end it should be necessary to define an order ($\prec$) between the $PerfC_iS_j\ (a_1, \ldots, a_m)$ indices that depends on the adopted measurement framework (i.e., what kind of measurement we are interested in: mean, distribution, best/worst case) and on the execution environment and then select the components following this order.

Finally, taking into consideration issue 3, a trustworthy performance prediction methodology should consider the integrated use of models and measures to exploit the inherent advantages of both methods and to handle the complexity of component-based systems. Moreover, the process of performance analysis should be automated as best as possible to avoid errors and increase the efficiency of performance prediction. This applies both to the derivation of the performance model as well as the model solution. Another, often neglected, important aspect that should be included in the prediction methodology is the assessment of the *goodness* of a model, i.e., how close the model is to the real system. This involves a verification step ensuring that the model is correctly built and a validation step ensuring that the model produces results close to those observed in the real system.

Figures 3 and 4 show a foreseeable component-based software design environment with a built-in performance prediction tool combining both measurement and modeling techniques. As illustrated in figures 3 and 4, the final goal should be to have some automatic tool allowing performance prediction at design time, so the effort required at assemby time at the software designer is quite small.

To summarize, the area of performance prediction for component-based systems is extensive and more research is necessary to achieve a full integration of quantitative prediction techniques in the software development process. Several related research directions should cover, for example the design of component models allowing quality prediction and building of component technologies supporting quality prediction. Further research is needed to include also other quality attributes, such as reliability, safety or security in the software development process. Another interesting point that deserves

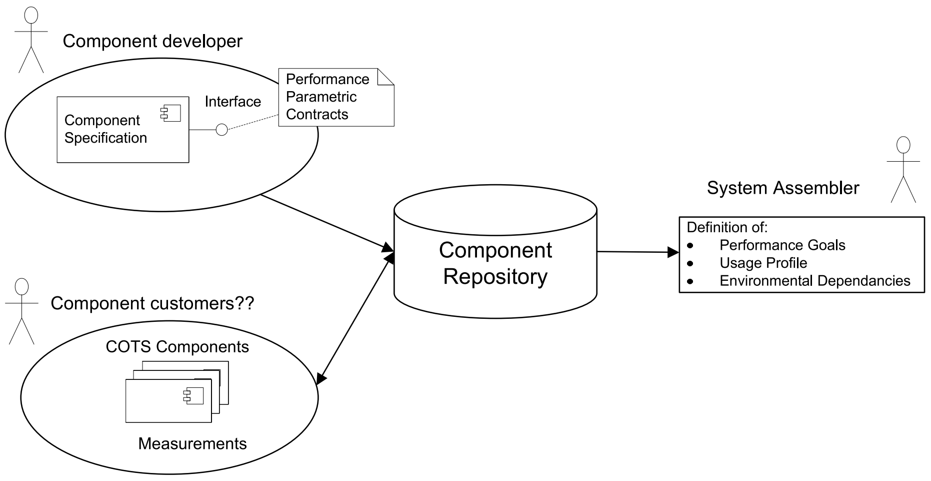
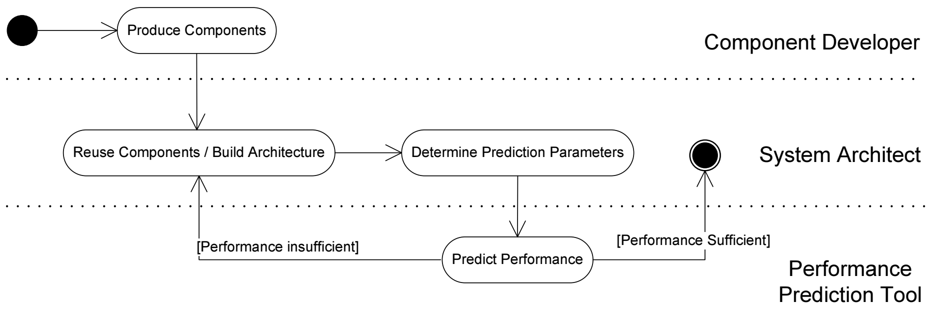


Fig. 3. *A CBSE Framework with Performance prediction included*



Produce Components
Component Developer
Reuse Components / Build Architecture
Determine Prediction Parameters
System Architect
[Performance insufficient]
Predict Performance
[Performance Sufficient]
Performance Prediction Tool

Fig. 4. *Activity diagram of the Performance Framework*

further investigation is to study interdependencies among the different quality attributes, to determine, for example, how the introduction of performance predictability can affect other attributes such as reliability or maintainability. To this end a good starting point would be a unified view on software quality attributes taking into account the various existing trade-offs between related quality attributes ([40]).

6 Conclusion

In this paper, we have reviewed performance prediction techniques for component-based software systems. Especially, we focus on the practicability of these techniques for various software engineering problem scenarios. For that reason, we have discussed some practical concerns that emerge when developing component-based systems and given an overview of current performance prediction approaches. As a result, we can present the inherent strengths and weaknesses of each performance prediction technique for different problems and system categories. This leads to a survey from an engineering perspective, which allows software engineers to select an appropriate performance prediction technique. Moreover, based on our examination of existing performance prediction techniques we have presented some recommendations for future research activities. Finally, we think that this survey could have a significant impact on the current software engineering practices and on the applicability of Component-Based Software Engineering methodologies.

Acknowledgements

The authors would like to thank Viktoria Firus and Ian Gorton for their valuable and inspiring input during our break-out session at Dagstuhl, which preceded this paper. Steffen Becker is funded by the German Science Foundation in the DFG-Palladio project.

References

1. Smith, C.U.: Performance Engineering of Software Systems. Addison-Wesley, Reading, MA, USA (1990)
2. Smith, C.U., Williams, L.G.: Performance Solutions: a practical guide to creating responsive, scalable software. Addison-Wesley (2002)
3. Szyperski, C., Gruntz, D., Murer, S.: Component Software: Beyond Object-Oriented Programming. 2 edn. ACM Press and Addison-Wesley, New York, NY (2002)
4. Sha, L., Rajkumar, R., Lehoczky, J.P.: Priority inheritance protocols: An approach to real-time synchronization. IEEE Trans. Comput. **39** (1990) 1175–1185
5. Musa, J.D., Iannino, A., Okumoto, K.: Software Reliability – Measurement, prediction, application. McGraw-Hill, New York (1987)
6. Dongarra, J., Martin, J., Vorlton, J.: Computer benchmarking: paths and pitfalls. IEEE Spectr. **24** (1987) 38–43
7. Balsamo, S., Marco, A.D., Inverardi, P., Simeoni, M.: Model-Based Performance Prediction in Software Development: A Survey. IEEE Transactions on Software Engineering **30** (2004) 295–310
8. Yacoub, S.M.: Performance Analysis of Component-Based Applications. In Chastek, G.J., ed.: Software Product Lines, Second International Conference, SPLC 2, San Diego, CA, USA, August 19-22, 2002, Proceedings. Volume 2379 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer (2002) 299–315
9. Chen, S., Gorton, I., Liu, A., Liu, Y.: Performance Prediction of COTS Component-Based Enterprise Applications. In: Proceedings of 5th ICSE workshop on Component-Based Software Engineering (CBSE 2002). (2002)
10. Denaro, G., Polini, A., Emmerich, W.: Early Performance Testing of Distributed Software Applications. In Dujmovic, J.J., Almeida, V.A.F., Lea, D., eds.: Proceedings of the Fourth International Workshop on Software and Performance, WOSP 2004, Redwood Shores, California, USA, January 14-16, 2004, New York, NY, ACM Press (2004) 94–103
11. Sitaraman, M., Kulczycki, G., Krone, J., Ogden, W.F., Reddy, A.L.N.: Performance Specification of Software Components. In: Proceedings of the Symposium on Software Reusability: Putting Software Reuse in Context, May 18-20, 2001, Toronto, Ontario, Canada, New York, NY, ACM Press (2001) 3–10

12. Hissam, S.A., Moreno, G.A., Stafford, J.A., Wallnau, K.C.: Packaging Predictable Assembly. In Bishop, J.M., ed.: Component Deployment, IFIP/ACM Working Conference, CD 2002, Berlin, Germany, June 20-21, 2002, Proceedings. Volume 2370 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer (2002) 108–124
13. Hissam, S., Hudak, J., Ivers, J., Klein, M., Larsson, M., Moreno, G., Northrop, L., Plakosh, D., Stafford, J., Wallnau, K., Wood, W.: Predictable Assembly of Substation Automation Systems: An Experiment Report. Technical Report CMU/SEI-2002-TR-031, Software Engineering Institute (2002)
14. Wallnau, K.C.: A Technology for Predictable Assembly from Certifiable Components. Technical Report CMU/SEI-2003-TR-009, Software Engineering Institute (2003)
15. Bertolino, A., Mirandola, R.: Towards Component-Based Ssoftware Performance Engineering. In: Proceedings of 6th ICSE workshop on Component-Based Software Engineering (CBSE 2003). (2003)
16. Bertolino, A., Mirandola, R.: CB-SPE Tool: Putting Component-Based Performance Engineering into Practice. In Crnkovic, I., Stafford, J.A., Schmidt, H.W., Wallnau, K.C., eds.: Component-Based Software Engineering, 7th International Symposium, CBSE 2004, Edinburgh, UK, May 24-25, 2004, Proceedings. Volume 3054 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer (2004) 233–248
17. OMG: UML Profile for Schedulability, Performance, and Time. OMG Specification ptc/2002-03-02, Object Management Group (2002)
18. Wu, X., McMullan, D., Woodside, M.: Component-Based Performance Prediction. In: Proceedings of 6th ICSE workshop on Component-Based Software Engineering (CBSE 2003). (2003)
19. Wu, X., Woodside, C.M.: Performance Modeling from Software Components. In Dujmovic, J.J., Almeida, V.A.F., Lea, D., eds.: Proceedings of the Fourth International Workshop on Software and Performance, WOSP 2004, Redwood Shores, California, USA, January 14-16, 2004, New York, NY, ACM Press (2004) 290–301
20. Balsamo, S., Marzolla, M.: A Simulation-Based Approach to Software Performance Modeling. In: ESEC/FSE-11: Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering, New York, NY, ACM Press (2003) 363–366
21. Eskenazi, E.M., Fioukov, A.V., Hammer, D.K., Obbink, H., Pronk, B.: Analysis and Prediction of Performance for Evolving Architectures. In IEEE, ed.: Proceedings of the 30th EUROMICRO Conference 2004, 31 August - 3 September 2004, Rennes, France, Los Alamitos, CA, IEEE Computer Society Press (2004) 22–31
22. Eskenazi, E.M., Fioukov, A.V., Hammer, D.K.: Performance Prediction for Component Compositions. In Crnkovic, I., Stafford, J.A., Schmidt, H.W., Wallnau, K.C., eds.: Component-Based Software Engineering, 7th International Symposium, CBSE 2004, Edinburgh, UK, May 24-25, 2004, Proceedings. Volume 3054 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer (2004) 280–293
23. Bondarev, E., de With, P.H., Chaudron, M.R.V.: Towards Predicting Real-Time Properties of a Component Assembly. In IEEE, ed.: Proceedings of the 30th EUROMICRO Conference 2004, 31 August - 3 September 2004, Rennes, France, Los Alamitos, CA, IEEE Computer Society Press (2004) 601–610
24. Reussner, R.H., Firus, V., Becker, S.: Parametric Performance Contracts for Software Components and their Compositionality. In Weck, W., Bosch, J., Szyperski, C., eds.: Proceedings of the 9th International Workshop on Component-Oriented Programming (WCOP 04). (2004)
25. Menasce, D.A., Ruan, H., Gomaa, H.: A Framework for QoS-Aware Software Components. In: Proceedings of the Fourth International Workshop on Software and Performance, WOSP

2004, Redwood Shores, California, USA, January 14-16, 2004, New York, NY, ACM Press (2004) 186–196

26. Diaconescu, A., Mos, A., Murphy, J.: Automatic Performance Management in Component Based Software Systems. In IEEE, ed.: Proceedings of the 1st International Conference on Autonomic Computing (ICAC 2004), 17-19 May 2004, New York, NY, USA, Los Alamitos, CA, IEEE Computer Society Press (2004) 214–221
27. Liu, Y., Fekete, A., Gorton, I.: Predicting the Performance of Middleware-Based Applications at the Design Level. In: Proceedings of the Fourth International Workshop on Software and Performance, WOSP 2004, Redwood Shores, California, USA, January 14-16, 2004, New York, NY, ACM Press (2004) 166–170
28. Wallnau, K.C., Stafford, J., Hissam, S., Klein, M.: On the Relationship of Software Architecture to Software Component Technology. In: Proceedings of the Sixth International Workshop on Component-Oriented Programming (WCOP'01), Budapest, Hungary, 19 June 2001. (2001)
29. Solberg, A., Husa, K.E., Aagedal, J.., Abrahamsen, E.: QoS-Aware MDA. In: Proceedings of the Workshop Model-Driven Architecture in the Specification, Implementation and Validation of Object-Oriented Embedded Systems (SIVOES-MDA'03) in conjunction with UML'03. (2003)
30. Grassi, V., Mirandola, R.: A Model-Driven Approach to Predictive Non-Functional Analysis of Component-Based Systems. In: Proceedings of the Workshop Models for Non-functional Aspects of Component-Based Software at UML 2004, 12 October 2004,. (2004)
31. Petriu, D.B., Woodside, C.M.: A Metamodel for Generating Performance Models from UML Designs. In Baar, T., Strohmeier, A., Moreira, A.M.D., Mellor, S.J., eds.: UML 2004 - The Unified Modelling Language: Modelling Languages and Applications. 7th International Conference, Lisbon, Portugal, October 11-15, 2004. Proceedings. Volume 3273 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer (2004) 41–53
32. Woodside, M., Petriu, D.C., Petriu, D.B., Shen, H., Israr, T., J.Merseguer: Performance by Unified Model Analysis (PUMA). In: Proceedings of the Fifth International Workshop on Software and Performance, WOSP 2005, Palma, Illes Balears, Spain, July 11-15, 2005, New York, NY, ACM Press (2005) forthcoming
33. Gu, G., Petriu, D.C.: From UML to LQN by XML Algebra-Based Graph Transformations. In: Proceedings of the Fifth International Workshop on Software and Performance, WOSP 2005, Palma, Illes Balears, Spain, July 11-15, 2005, New York, NY, ACM Press (2005) forthcoming
34. Grassi, V., Mirandola, R., Sabetta, A.: From Design to Analysis Models: A Kernel Language for Performance and Reliability Analysis of Component-Based Systems. In: Proceedings of the Fifth International Workshop on Software and Performance, WOSP 2005, Palma, Illes Balears, Spain, July 11-15, 2005, New York, NY, ACM Press (2005) forthcoming
35. Zschaler, S.: Towards a Semantic Framework for Non-Functional Specifications of Component-Based Systems. In IEEE, ed.: Proceedings of the 30th EUROMICRO Conference 2004, 31 August - 3 September 2004, Rennes, France, Los Alamitos, CA, IEEE Computer Society Press (2004) 92–99
36. Grassi, V., Mirandola, R.: Towards Automatic Compositional Performance Analysis of Component-Based Systems. In: Proceedings of the Fourth International Workshop on Software and Performance, WOSP 2004, Redwood Shores, California, USA, January 14-16, 2004, New York, NY, ACM Press (2004) 59–63
37. Frølund, S., Koistinen, J.: Quality-of-Service Specification in Distributed Object Systems. Technical Report HPL-98-159, Hewlett Packard, Software Technology Laboratory (1998)
38. Larsson, M.: Predicting Quality Attributes in Component-Based Software Systems. PhD thesis, Mlardalen University (2004)

39. Meyer, B.: Applying “Design by Contract”. IEEE Computer **25** (1992) 40–51
40. Crnkovic, I., Firus, V., Grunske, L., Jezequel, J.M., Overhage, S., Reussner, R.: Unified Models for Predicting the Quality of Component-Based Software Architectures. In Reussner, R., Stafford, J., Szyperski, C., eds.: Architecting System with Trustworthy Components. Lecture Notes in Computer Science, Berlin, Heidelberg, Springer (2005) forthcoming