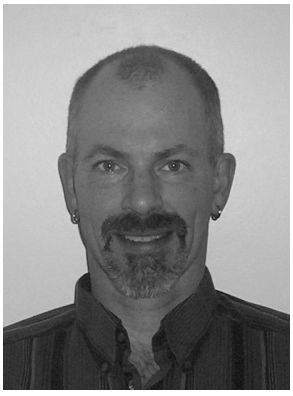
The Future of Software Performance Engineering

Murray Woodside, Greg Franks, Dorina C. Petriu

Murray Woodside obtained a PhD in Control Engineering in 1964 from Cambridge University, and has taught since 1970 at Carleton University, Ottawa. His research addresses performance and dependability of software. He has served as Vice-Chair and Chair of Sigmetrics, the ACM Special Interest Group on performance, and is an Associate Editor of Performance Evaluation. He is a Fellow of the IEEE.



Greg Franks is an Assistant Professor at the Department of Systems and Computer Engineering at Carleton University. His areas of interest are computer systems performance analysis, operating systems, IP routing, with emphasis on analytic performance modelling. He received the PhD degree from Carleton University, and has taught topics ranging from microprocessor interfacing to functional and imperative programming languages. He is a member of IEEE and ACM.

Dorina C. Petriu is a Professor in the Department of Systems and Computer Engineering at Carleton University, Ottawa, Canada. She received a Dipl. Eng. degree in computer engineering from the Polytechnic University of Timisoara, Romania, and a Ph.D. degree in electrical engineering from Carleton University. Her main research interests are in the areas of performance modelling and software engineering, with emphasis on integrating performance engineering into the software development process. She was a contributor to the "UML Profile for Schedulability, Performance and Time" (SPT) standardized by OMG, and is a member of an OMG working group defining the replacement of SPT. Dr. Petriu is a Fellow of the Engineering Institute of Canada, a Senior Member of I.E.E.E. and a member of A.C.M.

Future of Software Engineering(FOSE'07)
0-7695-2829-5/07 $20.00 © 2007 IEEE

The Future of Software Performance Engineering

Murray Woodside, Greg Franks, Dorina C. Petriu
Carleton University, Ottawa, Canada.
{cmw | greg | petriu}@sce.carleton.ca

Abstract

Performance is a pervasive quality of software systems; everything affects it, from the software itself to all underlying layers, such as operating system, middleware, hardware, communication networks, etc. Software Performance Engineering encompasses efforts to describe and improve performance, with two distinct approaches: an early-cycle predictive model-based approach, and a late-cycle measurement-based approach. Current progress and future trends within these two approaches are described, with a tendency (and a need) for them to converge, in order to cover the entire development cycle.

1. Introduction

Software performance (considered here as concerned with capacity and timeliness) is a pervasive quality difficult to understand, because it is affected by every aspect of the design, code, and execution environment. By conventional wisdom performance is a serious problem in a significant fraction of projects. It causes delays, cost overruns, failures on deployment, and even abandonment of projects, but such failures are seldom documented. A recent survey of information technology executives [15] found that half of them had encountered performance problems with at least 20% of the applications they deployed.

A highly disciplined approach known as Software Performance Engineering (SPE) is necessary to evaluate a system's performance, or to improve it. In this paper we propose the following SPE definition:

> ***Definition:*** *Software Performance Engineering (SPE) represents the entire collection of software engineering activities and related analyses used throughout the software development cycle, which are directed to meeting performance requirements.*

Two general approaches found in literature are discussed here, both under the SPE umbrella. The commonest approach is purely *measurement-based*; it applies testing, diagnosis and tuning late in the development cycle, when the system under development can be run and measured (see, e.g. [2][4][8][9]). The *model-based* approach, pioneered under the name of SPE by Smith [65][67] (see also [7] for a survey of modeling approaches), creates performance models early in the development cycle and uses quantitative results from these models to adjust the architecture and design with the purpose of meeting performance requirements. The SPE definition proposed in this paper is broader than the one given in [65][67], in that it also includes late-cycle measurement-based and model-based approaches.

An analogy with other engineering disciplines suggests that design by early calculations (e.g., by models) is the way forward, but also that it must be integrated with measurements. In the case of SPE, this integration has been elusive.

Like other software engineering activities, SPE is constrained by tight project schedules, poorly defined requirements, and over-optimism about meeting them. Nonetheless adequate performance is essential for product success, making SPE a foundation discipline in software practice.

Resources. A resource is a system element that offers services required by other system elements. Performance results from the interaction of system behaviour with resources, thus SPE gives first-class status to resources of all kinds. Resources include:

- hardware (CPU, bus, I/O and storage, network)
- logical resources (buffers, locks, semaphores)
- processing resources (processes, threads)

A determining factor for performance is that resources have a limited capacity, so they can potentially halt/delay the execution of competing users by *denying permission to proceed*. Quantifying such effects is an important task of SPE.

Full SPE capability implies bringing resources into models of software, as in the standard "UML Profile for Schedulability, Performance and Time" (SPT) [52] and its planned replacement "UML Profile for Modeling and Analysis of Real-Time and Embedded systems" (MARTE) [53]. Important properties of a resource are its multiplicity (units that can be assigned

Future of Software Engineering(FOSE'07)
0-7695-2829-5/07 $20.00 © 2007 IEEE

to requests, as in a buffer pool) and its scheduling discipline.

This paper introduces SPE domain and process concepts, surveys the current status of both approaches, and then considers movement towards convergence of measurement and modeling methods, into a single *Performance Knowledge Base*. This is the most prominent feature of our view of the future, but additional aspects of future development are also considered.

The survey of current work given here is only a sampling of papers and is far from complete.

2. SPE domain and process

The elements of the SPE domain are

- *system operations* (Use Cases) with performance requirements, behaviour and workloads,
- *behaviour*, defined by *scenarios* (e.g. by UML behaviour specifications),
- *workloads* (defining the frequency of initiation of different system operations),
- system *structure*, the software components
- *resources*, hardware and software.

Performance engineering normally confines itself to a subset of system operations (for which performance is of concern). These operations with their relative weight make up the *operational profile* (see, e.g. [2]).

Smith and Williams describe an SPE process using early modeling, based on significant experience [66]; Barber describes a process using measurement [9]. SPE processes are also discussed in [5][49][75]. In [77] a taxonomy of SPE processes was based on how they use three kinds of information: structural (called "maps"), behavioural ("paths") and resources.

2.1. SPE activities

Any SPE process is woven into software development and includes some or all of the following activities:

Identify concerns (important system operations and resources). The qualitative analysis of factors affecting performance goals is described in [14].

Define and analyze requirements: Define the operational profile, workload intensities, delay and throughput requirements, and scenarios describing behaviour. UML behaviour notation or special scenario languages are used (e.g. execution graphs [65], Use Case Maps [56], User Community Modeling Language (UCML) for performance test workloads [10]). The scenarios are the basis for designing performance tests [10], and for early performance models [57][66][56] [58][11].

Predict performance from scenarios, architecture, and detailed design, by modeling the interaction of the behaviour with the resources. Modeling techniques were surveyed in [6] This activity is discussed in Section 4 below.

Performance testing on part or all of system, under normal loads and stress loads [8]. The use of test data to solve problems is the subject of [9]. This activity is discussed in Section 3 below.

Maintenance and evolution: predict the effect of potential changes and additions. Examples include: impact of added features [64], impact of a platform migration [2], comparison of web application platforms [36].

Total system analysis: consider the planned software in the complete and final deployed system. The present work is related (and seamlessly connects) to this larger perspective, but concentrates on software development. An example of total analysis for future system planning is given in [60].

The SPE activities are summarized in Figure 1. The vertical placement indicates whether they are performed earlier or later in the software lifecycle: activities at the top correspond to early stages, at the bottom to late stages.

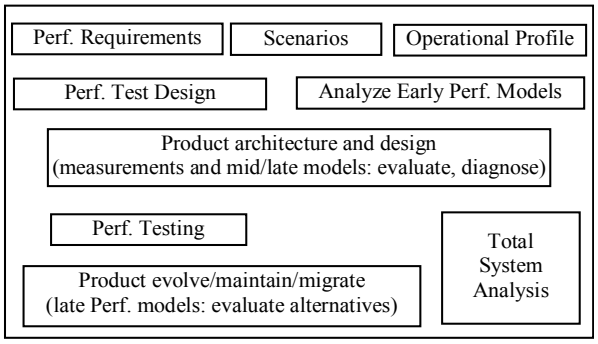


Figure 1 SPE activities

Lessons from the current work. There are many weaknesses in current performance processes. They require *heavy effort*, which limits what can be attempted. Measurements lack *standards*; those that record the application and execution context (e.g. the class of user) require source-code access and instrumentation and interfere with system operation. There is a *semantic gap* between performance concerns and functional concerns, which prevents many developers from addressing performance at all. For the same reason many developers *do not trust* or understand performance models, even if such models are available. Performance modeling is effective but it is often costly; *models are approximate*, they leave out detail that may be important, and are difficult to validate.

The survey of information technology executives [15], which found that half of them had had performance problems with at least 20% of the applications they deployed, commented that many problems seem to come from lack of coverage in performance testing, and from depending on customers to do performance testing in the field.

Significant process issues are unresolved. Detail in measurement and modeling must be managed and adapted. Excessive detail is expensive and creates information overload, while insufficient detail may miss the key factor in a problem. Information has to be thrown away. Models and measurements are discarded even though they possibly hold information of long-term value.

3. Progress in measurement, debugging and testing

Measurement is used by verification teams to ensure that the system under test meets its specifications, by performance modelers to build and validate models, and by designers to find and fix hot-spots in the code. Interest in the measurement of the performance of a computer system ranges back to the development of the very first systems, described in an early survey paper by Lucas [39]. Today, the state of industrial performance measurement and testing techniques is captured in a series of articles by Scott Barber [8][9] including the problems of planning, execution, instrumentation and interpretation.

For performance test design, an important issue is to determine the workload under which the testing is done. An approach is to run the performance tests under similar conditions with the expected operational profile of the application in the field [9]. Briand and co-workers have pioneered the use of models to create stress tests for time-critical systems, by triggering stimuli at strategic instants [22].

This section will first describe some of the tools and techniques used to measure the performance of a system. Next, problems that exist today which impede the adoption of the tools will be described. Finally, future directions are identified.

3.1. Tools

The tools used by performance analysts range from load generators, for supplying the workload to a system under test, to monitors, for gathering data as the system executes. Monitoring itself can be performed through hardware, software, or a combination of the two. The focus of this paper will be on software monitoring, which itself can be broken down into two broad categories: instrumentation and profiling.

Instrumentation is the insertion of probes into a system to measure some sort of events. Some instrumentation is usually built into the host operating system and minimally indicates the utilization of the various devices including the CPU. Other instrumentation is added manually to applications. Frameworks such as the Application Response Measurement (ARM) application programmer interface [29] are beneficial as they form a common substrate to which disparate programs can gather performance information.

Instrumentation can also be added automatically. Aspect-Oriented programming can be used to automatically insert instrumentation code into applications [19]. Quantify [26] adds probes to object code at the beginning and ending of basic blocks of programs to count the number of cycles executed. The Paradyn tool [47], carries this one step further, by instrumenting the actual executables dynamically.

Profilers. A program profile is a histogram of the execution of a program [31]. It can be generated using instrumentation, as is the case with Quantify, through statistical sampling, or by running the program on a virtual machine and counting the execution of the actual instructions [50].

3.2. Problems

The use of performance tools is well established at the verification level, where it used to ensure that QoS requirements are being met. However, it is less well established at earlier stages in the life-cycle of a product. Malony and Helm [42] have identified two obstacles in particular to the adoption of these tools

a) a lack of theoretical justification for the methods for improving performance that work, and why they do so. Tools will provide measurement data, but expert interpretation is still required to fix problems.
b) a conflict between automation and adaptability in that systems which are highly automated but are difficult to change, and vice versa. As a result no tool does the job the user needs, so the user goes and invents one. Further, various tools all have different forms of output which makes interoperability challenging at best.
c) in distributed systems, events from different systems need to be correlated. Today's systems are often composed of sub-systems from different vendors. Establishing causality across the system is difficult.

3.3. The future

Performance engineering is gaining attention, as companies discover to their detriment that the performance of their applications is often below expectations. In the past, these problems were not found until very late in the development of a product as performance validation, if any, was one of the last activities done before releasing the software. With agile processes, the problem is unchanged if not worse [11]. Thus early warning of performance problems is still the challenge for SPE.

Better tools for measurement and modeling are one direction we shall examine. Tracing captures CPU demands quite well, but operating system support is needed to trace operations to the various I/O devices. Distributed systems require correlating traces between nodes. The ARM framework [29] supports this capability today, but it is still up to developers to write the correlation code. Advances in logical clocks [25] can aid this work.

Developers and testers use instrumentation tools to help them find problems with systems. However, users depend on experience to use the results, and this experience needs to be codified and incorporated into tools. Better methods and tools for interpreting the results and diagnosing performance problems are a future goal.

To be successful, earlier performance diagnosis requires the use of models for insight into the sources of problems. Model calibration may rest on improved tracing technology. Rapid cheap modeling techniques are a desirable goal.

At present, there is very little standardization in file and model formats and protocols used in performance engineering. The ARM framework is making progress in terms of tracing, the Performance Model Interchange Format is being proposed for queueing models [68], and the UML Profile for Schedulability, Performance, and Time Specification [52] has been adopted for UML software design tools. Most tools, however, do not conform to these formats. Further, these formats themselves may not be sufficient to cover the measurement and analysis domain.

Workload modeling and operational profiles are a key part of both load testing and predictive modeling. A discussion of how critical a good workload model is, and how it can be constructed, is found in [2][33].

Dynamic optimization can use measurements fed back to compilers for tuning, for off-line placing code and other optimizations, and for cache analysis. Recently instrumentation was described for caches, to estimate the effect of an increment in the cache size from the application behaviour while it runs [76], and to control the cache size.

Performance models are often difficult to construct, even with a live system, despite the presence of tools to actually measure performance. In the future, model building will become much more automated, and output becomes standardized, and the conversion process between measurement information and performance model becomes more practical. Ultimately, the model and measurement information will be fed back into design tools, so that performance issues are brought to the forefront early in the design process.

4. Prediction of performance by models

Performance models describe how system operations use resources, and how resource contention affects operations. The types of models used for software, including queueing networks, layered queues, and types of Petri Nets and Stochastic Process Algebras, were surveyed recently by Balsamo et al [7].

The special capability of a model is prediction of properties of a system before it is built, or the effect of a change before it is carried out. This gives a special "early warning" role to early-cycle modeling during requirements analysis. However as implementation proceeds, better models can be created by other means, and may have additional uses, in particular

- design of performance tests
- configuration of products for delivery
- evaluation of planned evolutions of the design, recognizing that no system is ever final.

Incremental change: models are ideal for evaluating the performance impact of changes which can be implemented in a variety of ways.

Model validation: validation is critical for a model created to represent an existing system in detail. For a planned system, on the contrary, it is a non-issue. The model simply summarizes the designer's knowledge. Like a project budget, which also represents the future, it represents knowledge with uncertainty which is validated by experience in using it. Like a budget, it has an element of risk.

4.1. Performance models from scenarios

Early performance models are usually created from the intended behaviour of the system, expressed as scenarios which are realizations of Use Cases. The term "scenario" here denotes a complex behaviour including alternative paths as well as parallel paths and repetition. The performance model is created by extracting the demands for resource services. This has been described for different source scenario models and target performance formalisms:

- from Markov models of sequence, to queueing models [63][55]
- from execution graphs to queueing models [66][15]
- from Message Sequence Charts and SDL to simulations [30][49]
- from UML: as surveyed in [80]
- from team expertise, as described by Smith and Williams [66].

These works clearly show the feasibility of model creation. Important challenges remain however (1) in making the process accessible to designers, which is addressed by the SPT profile [52] (giving a software design context for the annotations); (2) in creating interoperability between design and performance tools (the model interchange language in [68] is a promising development in this direction); (3) in providing models for parts of the systems not described by the designer; and (4) in providing a scalable and robust model.

Annotated UML specifications are a promising development. The annotations include:

- the workload for each scenario, given by an arrival rate or by a population with a think time between requests,
- the CPU demand of steps,
- the probabilities of alternative paths, and loop counts,
- the association of resources to the steps either implicitly (by the processes and processors) or explicitly.

As an illustration, Figure 2 shows a set of applications requesting service from a pool of server threads running on a multiprocessor (deployment not shown). Part (a) shows the scenario modeled as a UML sequence diagram with SPT annotations, (b) shows a graph representing the scenario steps, and (c) shows the corresponding layered queueing network (LQN) model. Studies in [44] [56] [58] [72] use such models.

At a later stage, scenarios may be traced from execution of prototypes or full deployments, giving accurate behaviour. Models can be rebuilt based on these experimental scenarios [24][78], combined with measured values of CPU demands.

Schedulability analysis of time-critical systems [32] is a special kind of scenario-based analysis, which is beyond the scope of this paper.

4.2. Performance models from objects and components

A different approach can be used when the architecture is in place. A performance model can be built based on the software objects viewed from a performance perspective. A pioneering contribution in this direction defined a "performance abstract data type" for an object [13], based on the machine cycles executed by its methods.

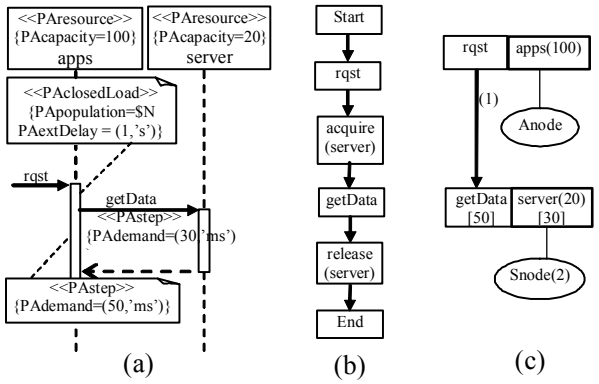


Figure 2. Annotated UML, Scenario Model, and Performance Model

To create a performance model, one traces a response from initiation at a root object to all the interfaces it calls, proceeding recursively for each call. Queueing and layered queueing models were derived based on objects and calls in [55] [44] and [83]. Model parameters (CPU, call frequencies) were estimated by measurement or were based on the documentation plus expertise.

Object-based modeling is inherently compositional, based on the call frequencies between objects. This extends to subsystems composed of objects, with calls between subsystems. In [2] an existing application is described in terms of UNIX calls, and its migration to a new platform is evaluated by a synthetic benchmark with these calls, on the new platform. This study created a kind of object model, but then carried out composition and evaluation in the measurement domain. The convergence of models and measurements is an important direction for SPE.

The object-based approach to performance modeling can be extended to systems built with reusable components. Composition of submodels for Component-Based Software Engineering [71] was described in [11] [36] [59] [62] [82]. Issues regarding performance contracts between components are discussed in [59]. Components or platform layers can be modeled separately, and composed by specifying the calls between them. For example, in [36] a model of a J2EE application server is created as a component that offers a large set of operations; then an application is modeled (by a scenario analysis) in terms of the number of calls it made to each operation.

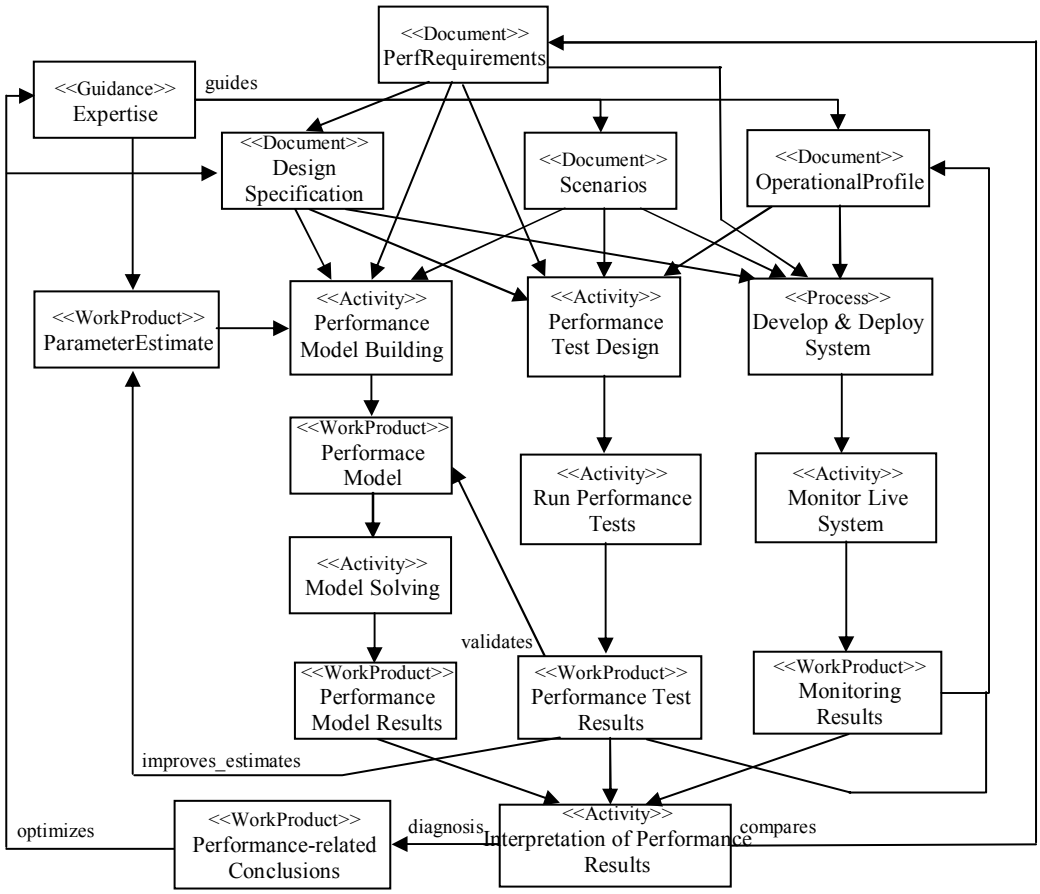


Figure 3. Simplified domain model for a converged SPE process

The quantitative parameters of the performance model for the J2EE server - and the underlying operating system and hardware platform - were obtained by measurements for two different implementations.

The main challenge regarding performance characterization of reusable components stem from the fact that the offered performance depends not only on the component per se, but also on its context, deployment, usage and load. It seems obvious that such approaches apply similarly to Generative techniques [17] and to Model-Driven Development; this point is followed up in Section 6.

The *completion* of performance models made from a software design, by adding components that make up its environment but are outside the design, is also largely based on composition of submodels [79]. This is an aspect of Model-Driven Development, addressed below.

5. Convergence of the measurement and modeling approaches

The present state of performance engineering is not very satisfactory, and better methods would be welcome to all. One way forward is to combine knowledge of different kinds and from different sources into a converged process. Figure 3 outlines such a process, with the main concepts and their relationships. The notation is based on the newly adopted OMG standard Software Process Engineering Metamodel (SPEM) [54]. At the core of SPEM is the idea that a software process is a collaboration between abstract active entities called *ProcessRoles* (e.g., use-case actors) that perform operations called *Activities* on concrete entities called *WorkProducts*. Documents, models, and data are examples of WorkProduct specializations. Guidance elements may be associated to different model elements to provide extra information. Figure 3 uses stereotypes defined in [54]. Concepts related to the model-based approach appear

on the left of Figure 3, and to the measurement-based approach on the right. A distinction is made between performance testing measurements (which may take place in a laboratory setting, with more sophisticated measurement tools and special code instrumentation) and measurements for monitoring live production systems that are deployed on the intended target system and used by the intended customers. The domain model from Figure 3 is very generic. For instance, there is no indication whether different activities (such as Performance Model Building) are done automatically through model transformations or "by hand" by a performance analyst. Some of these aspects will be discussed in the following sectionsIn a convergence of data-centric and model-centric methods, data (including prior estimates) provides the facts and models provide structure to organize and to extract significance from the facts. Our exploration of the future will examine aspects of this convergence. Models have a key role. They integrate data and convert it from a set of snapshots into a process capable of extrapolation. To achieve this potential we must develop robust and usable means to go from data to model (i.e., model-building) and from model to "data" (solving to obtain predictions). We must also learn how to combine measurement data interpretation with model interpretation, and to get the most out of both.

.Capabilities supported by convergence include:

- efficient testing, through model-assisted test design and evaluation
- search for performance-related bugs,
- performance optimization of the design
- scalability analysis
- reduced performance risk when adding new features,
- aids to marketing and deployment of products.

The future developments that will provide these capabilities are addressed in the remainder of this section. A future tool suite is sketched in Figure 4.

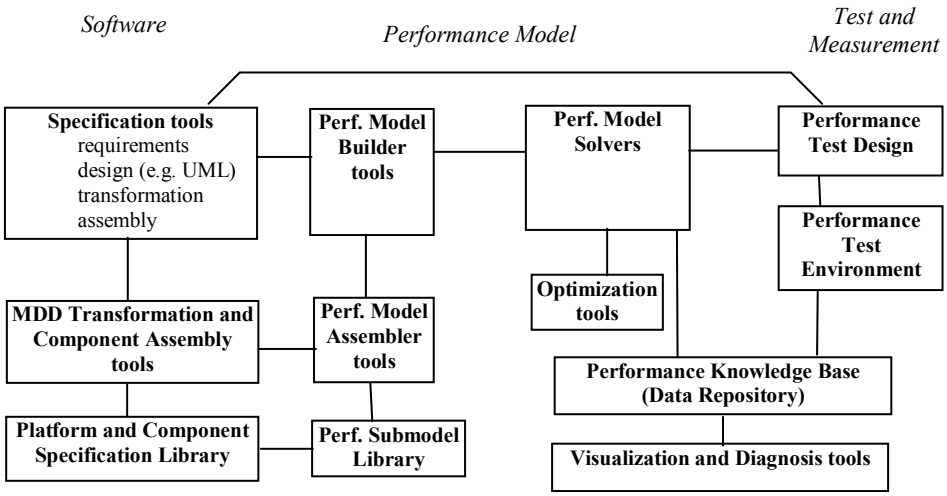


Figure 4: Tools for a Future Converged SPE Process, linked to Software Development Tools

6. Technical developments

Future technical developments in SPE can be placed within a converged process, and illustrate the opportunities offered by convergence of measurement and model based thinking.

6.1. Visualization and diagnostics

Understanding the source of performance limitations is a search process, which depends on patterns and relationships in performance data, often revealed by visualizations. Promising areas for the future include better visualizations, deep catalogues of performance-related patterns of behaviour and structure, and algorithms for automated search and diagnosis.

Present visualization approaches use generic data-exploration views such as Kiviat graphs (e.g. in Paradyn [47]), traffic intensity patterns overlaid on physical structure maps [47], CPU loading overlaid on scenarios, and breakdowns of delay [66]. Innovative views are possible. For example, in [60] all kinds of resources (not just the CPU) are monitored, with tools to group resources and focus the view. The challenge for the future is to visualize the causal interaction of

behaviour and resources, rather than to focus on just one or the other.

Bottleneck identification, a search for a saturated resource which limits the system, is a frequent operation. In [21], Franks et al describe a search strategy over a model, guided by its structure and results, and detects under-provisioned resource pools and over-long holding times. It combines properties of resources and behaviour, for a "nested" use of resources. It scales to high complexity, is capable of partial automation, and could be adapted to interpret measured data. A multistep performance enhancement study using these principles is described in [83]. Another search strategy purely over data ([9], part 7) focuses on reproducing and simplifying the conditions in which the problem is observed. The actual diagnosis of the cause (e.g. a memory leak) depends on designer expertise.

A bottleneck search strategy combining the data and the model could detect more kinds of problems (e.g. both memory leaks and resource problems) and could provide automated search assistance.

Patterns (or anti-patterns) related to bottlenecks have been described by Smith and Williams [66] and others (e.g., excessive dynamic allocation, "one-lane bridge"). For the future, more patterns and more kinds of patterns (on measurements, on scenarios or traces) will be important. Patterns that combine design, model and measurement will be more powerful than those based on a single source. A sketch of such a "super-pattern" for bottleneck discovery is as follows:

Bottleneck pattern: Resource R is a candidate bottleneck if:

1. it is used by the majority of scenarios,
2. many scenarios that use it are too slow,
3. it is near saturation (>80% of its units are busy),
4. resources that are acquired earlier and released later are also near saturation (from [21]).

Candidates can be resolved by problem-solving strategies, and by probing with more measurements.

Automated assistance to diagnose and even correct bottleneck problems could give (for the system level) the capability of optimizing compilers. This might be part of a high-level process such as MDD, (Section 6), and will depend on combining model-level abstractions with data.

Scalability analysis and improvement is largely a matter of identifying bottlenecks that emerge as scale is increased, and evolving the architecture. Future scalability tools will employ advanced bottleneck analysis but will depend more heavily on models, since they deal with potential systems.

6.2. Model-based knowledge integration

The knowledge created during performance studies tends to be fragmented and lost. Great value could be obtained by organizing and retaining it in a *"Performance Knowledge Base"* shown in Figure 4, including performance data and model predictions across time and across system versions. Such a knowledge base could be organized around a generalized notion of a performance model. This model is more than just a calculation to predict performance values. It is an abstraction created to support reasoning and exploration, and it describes the system with different values of *factors* that govern and modify performance. Factors can include variations in design, workload, component combinations, configuration, and deployment.

Every system exists in many actual and potential versions, over time and across alternative designs and configurations. The workload and behaviour depend on the context of operation (scale of deployment, domain of use, time of day...). One model exists as a set of cases, with pointers to trace each case back to the factors which define it. Factors include assumptions and expectations, specification documents, scenarios, performance measures and requirements, test data, intended deployment and context of operation. The mathematical definition of the predictive function will vary from case to case, both in parameters and in structure, and the model should document how these were derived from the governing factors.

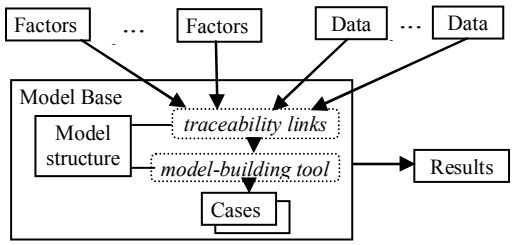


Figure 5. Modeling on demand and the model base

To make modeling maintainable and in sync with the current state of a system, it should be possible to construct a model case on demand, from the current state of the factors. The recipe for doing this is a "*Model Base*", consisting of model structures with pointers to sources of parameters, and rules to govern the creation of the appropriate model for the case desired. The Model Base, factors, parameter data, and rules are all elements of the Performance Knowledge

Base, and the calculator aspect of the model is kind of a projection of it. Model-building tools have a key role in such an approach.

6.3. Begin with requirements

Requirements analysis for performance can be enhanced by greater use of models; we can see the birth of the necessary tools, especially in the UML SPT Profile [52] for annotating specifications, and tools to build models from it [6] [37] [58] [80] [82]. Uses include specifying acceptance tests, judging the feasibility of a requirement, tradeoff analysis [14], and well-formedness (a search yielded no references on this last item, yet anecdotal evidence for it is widespread).

6.4. More efficient testing

Efficient testing covers the operational profile and the resources of interest with minimum effort needed to give sufficient accuracy. Accuracy is an issue because statistical results may require long runs, and it can be affected by other factors in the test design such as the load intensity and the patterns of request classes used. For example, systems under heavy load show high variance in their measurements, which contributes to inaccurate statistical results. It may be more fruitful to identify the heavily-used resources at moderate loads and (for purposes apart from stress testing) use the results to extrapolate to heavy loads using a performance model.

A model could assist in test design to ensure that all resources are used in some tests (resource coverage) and that effort is not wasted by running many experiments in the same saturation regime. At present coverage measures are not well-developed for performance tests. The model could be used to scale workloads, to select test configurations which are sensitive to system-features of interest, and to calibrate stub services with timing features.

More effort is also required in performance testing tools. The lack of standards for tool interoperation increases the effort to gather and interpret data, and reduces data-availability for new platforms. Lightweight and automated instrumentation are old goals that will continue to demand attention. Load drivers are at present well-developed in commercial tools, but more open tool development could speed progress.

6.5. Better models and solvers: goals

Performance models will be improved as regards capability (e.g., sensitivity analysis, scalability, models of time-varying systems), and numerical methods. Models depend on assumptions, and the sensitivity of the results to the assumptions is sometimes not known. For instance, an assumption of exponential service demand is only occasionally sensitive, (e.g., in finding the probability of timeout). Future models should provide built-in sensitivity calculations and warnings.

Analytic performance models based directly on states and transitions do not (yet) scale well due to state space explosion. Recent progress in numerical solution methods will continue, including better approximations. Models based on queues scale better, but use queueing approximations which need to be improved, notably for priorities, and for new scheduling disciplines such as fair sharing. Time-varying features due to mobility or to adaptation (in the run-time or in system management) will require new kinds of models and approximations.

Analytic model solvers sometimes fail due to bad numerical properties. By analogy with the handling of stiff systems of equations, there might be automatic fallback to a model that does not have such a problem.

Simulation modeling is steadily becoming more practical with the availability of more powerful inexpensive computers. However simulation model-building is still expensive, sometimes comparable to system development, and detailed simulation models can take nearly as long to run as the system. Simulation would be enhanced by automated model-building that includes significant abstraction (and thus reduced run-times) [6].

An alternative to simulation is a kind of simplified prototype, possibly combined with simulated components. This has been explored (e.g., [5][30]) but not perfected.

6.6. Efficient model-building tools

The abstractions provided by performance models are valuable, but some way must be found to create the models more easily and more quickly.

For performance models made early in the lifecycle from specified scenarios, automated model-building has been demonstrated [6] [37] [58] [80] [82] and is supported by the UML profiles [52] [53]. The future challenge is to handle every scenario that a software engineer may need to describe, and every way that the engineer can express them (including the implied scenario behaviour of object call hierarchies, and the composition of models from component designs).

The multiplicity of model formats hinders tool development, and would be aided by standards for performance model representations, perhaps building on [69]. Interoperability of performance building tools with standard UML tools is also helpful. For instance,

the PUMA architecture[80] shown in Figure 6 supports the generation of different kinds of performance models (queueing networks, layered queueing networks, Petri nets, etc.) from different versions of UML (e.g., 1.4 and 2.0) and different behavioural representations (sequence and activity diagrams). PUMA also provides a feedback path for design analysis and optimization.

Mid and late-cycle performance models should be extracted from prototypes and implementations. Trace-based automated modeling has been described in [24][27], including calibrated CPU demands for operations [78]. Future research can enhance this with use of additional instrumentation (e.g. CPU demands, code context), efficient processing, and perhaps exploit different levels of abstraction. Abstraction from traces exploits certain patterns in the trace, and domain-based assumptions; these can be extended in future research.

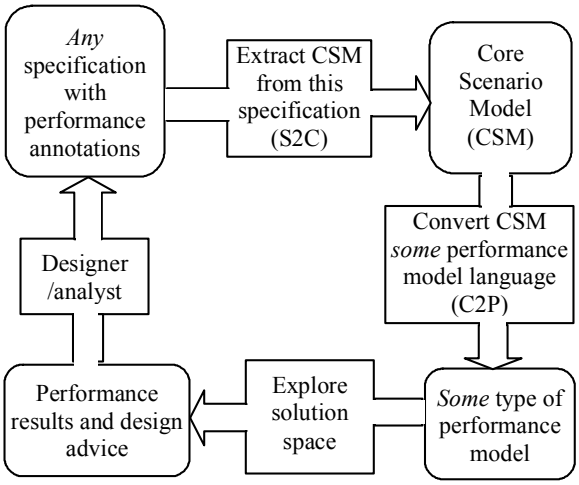


Figure 6 Architecture of the PUMA toolset [80]

7. Software performance in the context of Model Driven Development

Model-Driven Development (MDD) is an evolutionary step in software engineering that changes the focus of software development from code to models, as described in [20]. The Object Management Group (OMG) uses the copyrighted term Model-Driven Architecture (MDA) [51] to describe its initiative of producing standards for MDD (such as UML, MOF, XMI and CWM). MDD would be enhanced by the ability to analyze non-functional properties of software models [17]. This section discusses SPE research challenges in the context of MDD.

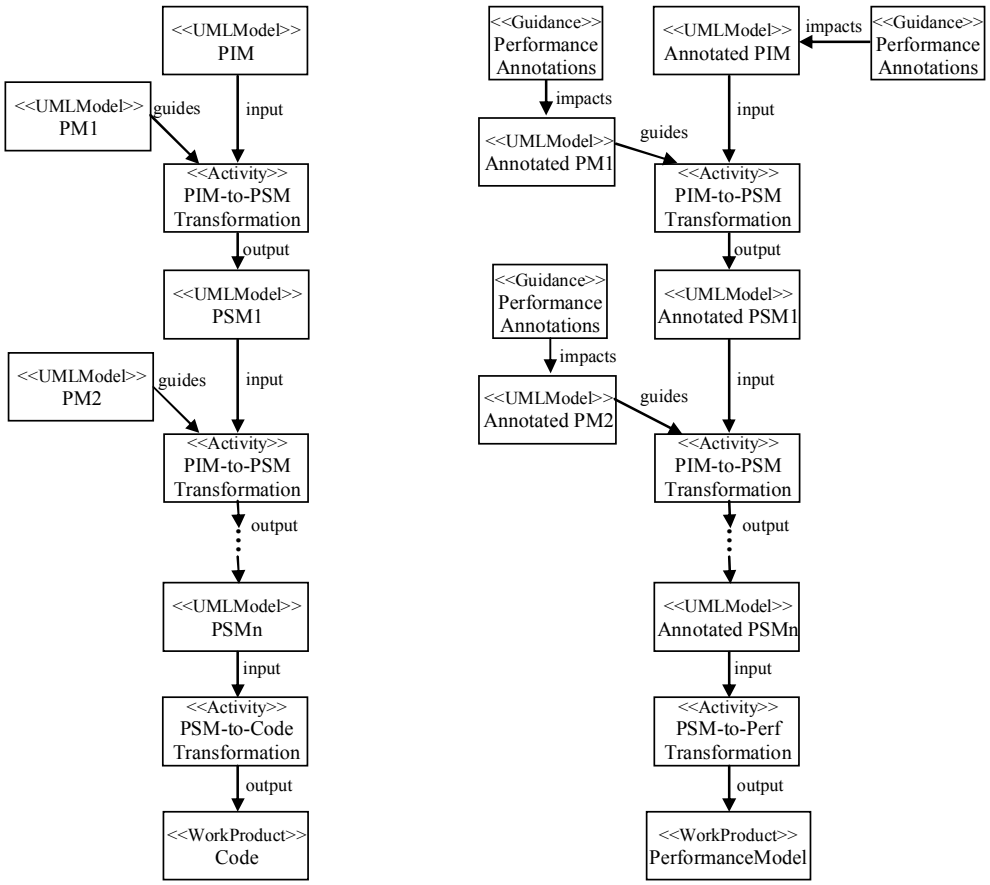
MDD is based on abstraction and automation: abstraction separates the model of the application under construction from the underlying platforms, and automation is used to generate the code from models. A key notion in MDA is that of "platform independence": the business and application logic of the software under development is represented separately from the underlying platform technology by a so-called Platform Independent Model (PIM). The term "platform" refers to technological and engineering details that are irrelevant to the fundamental functionality of the application. An application is usually supported by a number of layered platforms (such as virtual machine, operating system, middleware, communication software, hardware, network), each one contributing to the overall performance of the system. Platform models are usually layered, each providing services to the layers above, and requiring services from the layers below. The application model corresponding to a given platform, named Platform-Specific Model, is obtained by transforming the application PIM, as shown in Figure 7(a); the transformation is guided by information about the respective platform, given as a Platform Model (PM). Figure 7(a) illustrates a chain of PIM-to-PSM transformations (one for each layered platform). Since PSM_1 obtained for platform *i=1* is independent of its underlying platform *2*, therefore it can be denoted as PIM_2. Similarly for any *i=1* to n, PSM_i can also be denoted as PIM_{i+1}. The last transformation in the chain generates code from the last PSM.

One of the SPE goals in the context of MDD is to generate an overall performance model from a set of input models: an application PIM with performance annotations, plus a set of PMs for the underlying platforms, which should come with reusable performance annotations.

The purpose of the SPE model transformation chain shown in Figure 7(b) is different from the traditional MDD, as the target is a performance model rather than code. The SPE transformations can be approached in two ways: (i) follow the transformation chain from PIM to the last PSM in the software domain, then transform the last PSM into a performance model, as in Figure 7(b), or (ii) transform all input UML models into corresponding performance model fragments, then compose them into an overall performance model (the last composition taking place in the performance domain). In both cases, the transformation has to deal with the composition of performance annotations. The advantage of integrating the SPE model transformations into the MDD process is that software artifacts and performance models will be kept coherent and synchronized throughout the development process; this will simplify considerably the SPE activities.

The platform models do not have to be detailed and complete, as their role is to provide guidance for the PIM to PSM transformation rather than to give a full description of the platform details. Most probably, each platform should be characterized by the services it provides. The question is what abstraction level is enough. It is desirable to capture the most relevant performance characteristics of each platform, without providing unnecessary details. Also, it is possible that the PM abstraction level required for code generation is different from the level required for performance analysis. Since MDD is based on the platform-independence concept, it would be desirable to use separation of concerns when describing the performance contributions of each platform to the overall system performance. Thus, another SPE research challenge in the context of MDD is to define performance annotations for a given platform that are: a) reusable, and b) separated form the other platforms For instance, it would be useful to describe separately the performance contribution of a J2EE platform from the underlying operating system and hardware platforms. This could be achieved by using parametric annotations that express resource demand in one layer as functions of the services offered by the immediate underlying layer. However, with such an approach, vertical dependencies between platforms that are not immediate neighbours may be lost.

Complex software systems are usually built from components. Even though component-based development is not limited to MDD, it is important to address the question of reusable components in this context. The novelty here is to use component PIMs that will be composed into an application PIM. This constitutes a "horizontal" composition at the application level, as opposed to a "vertical" composition of an application with its supporting platforms.



a) MDD transformation from PIM to PSM for multiple layered platforms

b) SPE in the context of MDD: performance model generation through model transformations

Figure 7. MDD model transformations for generating (a) code, and (b) performance model

8. Optimization

A prime goal of future work is automatic performance optimization of architecture, design and run-time configuration. We see optimization as an extension of tuning, to include more fundamental changes to the system.

8.1. Design optimization

A first approach uses methods, not yet well worked out, to optimize design decisions represented in performance models. Manual search within this approach is described in [66] and [83]. An *evolutionary* approach could apply automated search techniques to models derived from the specification, with results fed back into the UML. Alternatively the search could be carried out on the UML design, with evaluation by models. The design can be parameterized, so optimization can be applied to the parameter values.

A second *constructive* approach is to begin from the requirements in the form of scenarios, and proceed to optimally cluster operations and data into objects and concurrent processes, also selecting communications patterns subject to various constraints (such as mutual exclusion).

Both approaches can be adapted to re-use of component and platform services with known properties, applying optimization to the transformation steps which add these features. In both cases also there are design decisions (such as storage mapping) that are below the level of abstraction in the model, which will perhaps be evaluated and optimized by running small measurement experiments on prototypes, or by applying general rules based on the context.

Both approaches also require some way to provide defaults for the configuration decisions discussed next, since performance can only be evaluated for a complete system.

8.2. Configuration optimization

Configuration decisions are applied at load and run time to adapt the software to its workload, platform and environment. Examples of decisions needing optimization include replication levels of servers and data [28], allocation of distributed processes to processors [32], to priorities [85]. Further candidates include buffer and thread pools [83], [43], middleware parameters [23], and virtual machine parameters. The future is expected to include systematic and efficient optimization approaches covering all these problems.

For complex products, optimization problems yet to be solved include selection of alternative components from a product line, and selection of their configuration parameters and installation parameters.

In [70] is presented an experimental methodology to evaluate the statistical significance of configurable parameters in e-commerce systems. Measurements are used to identify key configuration parameters that have a strong impact on performance, which are ranked in order of relevance. The impact of various parameters on the types of requests submitted to the site is also analyzed.

A different experimental approach to detect performance degradation in software systems with large configuration spaces is proposed in [84]. Formally designed experiments, called *screening designs*, are run with the goal of identifying important performance effects caused by the simultaneous interaction of n configuration parameters ($n = 1,2,...$). The conclusion was that screening designs can correctly identify important options used to produce reliable performance estimates across the entire configuration space, at a fraction of the cost of exhaustive testing.

In general, such experimental methods require a lot of computing power, so heuristics may be necessary to select the most promising experiments.

9. Additional Concerns

Agile Programming is becoming a mainstream approach to development, in which very frequent releases are made with small increments in function. Ancillary documents tend to be ignored, to concentrate on code. Quality control in agile programming is a current issue, for instance the role of test cases, and performance engineering fits into this concern. Performance specifications may not even be known for each incremental release.

Given the importance of performance to final products, a way must be found to do SPE in agile development, as discussed in [11]. This can be through test procedures or through models based on suitable design documents (e.g. based on Agile Modeling, e.g. [44]), or on test executions. The Performance Knowledge Base proposed here may be well-adapted to agile development, because it can be driven from the code (through tests and automatically generated models) and it accumulates history that is useful.

Design for Evaluation, Predictability and Adaptation: Progress can be expected in software design ideas that make it easier to evaluate and manage performance, something like design for testability.

A framework called PACC, Predictive Assembly from Certifiable Components [47], addresses the predictability of properties (including performance) of

systems built from components, applied to real-time system assembly. It shows that some properties of some kinds of components can be predicted with certainty. This is a question with wide practical resonance.

Adaptive or autonomic systems make performance engineering decisions on the fly, by optimizing the configuration of a running system. Software design for adaptation must include reflective capabilities to monitor its performance, strategies for adaptation, and handles on the system for effecting the necessary changes. Design considerations in adaptive computing are considered in [32].

Some strategies for adaptation are based on simple rules and simply modify provisioning of processors, but others use performance models as a basis for decisions [1]. Since the system may be changing, the use of models and tracking of model parameters for autonomic control was addressed in [81]. These models can be derived during development.

Integration with other evaluations: Other "non-functional" properties of software systems may also be evaluated from design documents, and it is natural to look ahead to integrating them with performance evaluation. For example security mechanisms often have a heavy performance penalty, and a joint evaluation would support a tradeoff between security effectiveness and performance cost. Reliability/ availability concerns have long been integrated with performance analysis in "performability" modeling (e.g. [72]), for similar reasons.

Ensuring system performance over the life of the software brings in concerns which are usually thought of as capacity planning, deployment or configuration, and management. The boundaries are not sharp, for instance capacity planning affects requirements for the software, and configuration must be considered in evaluation as described just above.

Scalability: this is a complex issue that rests heavily on performance limitations and bottlenecks. Research into scalable design approaches and patterns would be fruitful for the field.

10. Conclusions

Software Performance Engineering needs further development in order to cope with market requirements and with changes in software technology. It needs strengthening in prediction, testing and measurement technology, and in higher-level techniques for reasoning and for optimization.

The authors' main conclusion is that progress requires a convergence of approaches based only on measurement, with those that exploit performance models. The essential role of models is to integrate partial views, to extrapolate results from measured data, and to explore increments in the design.

Several areas of technical improvement in SPE have been identified for early progress. Further automation of data collection and better methods for deducing causality look promising. More powerful and general approaches to problem diagnosis are necessary and possible. Better methods for deriving and updating models are needed for the convergence above. Composition and transformation of models will support component-based systems and MDD, and performance optimization will help to exploit the models we can build.

Performance knowledge tends to be fragmented, and to be quickly lost. The authors propose a Performance Knowledge Base to integrate different forms of data which were obtained at different times, to support relationships between them, and to manage the data over time, configurations and software versions. This Performance Knowledge Base could act as a data repository and also contain results of analyses. It could be searched by designers and by applications to find relevant data for particular SPE purposes; we could see it as an application of Data Mining to software performance. It is suggested here that the Knowledge Base should be organized around performance model concepts.

SPE extends out from development into system deployment and management, and these efforts can also be integrated into the Knowledge Base, from which they can feed back into development increments.

Acknoweldgements

The comments of Scott Barber, Vittorio Cortellessa and Jerry Rolia are gratefully acknowledged. This research was supported by the Strategic Grants program of NSERC, the Natural Sciences and Engineering Research Council of Canada.

11. References

[1] B. Abrahao, V. Almeida, J. Almeida, A. Zhang, D. Beyer, F. Safai, "Self-Adaptive SLA-Driven Capacity Management for Internet Services", *Proc. 10th IEEE/IFIP Net Operations and Management Symp.*, 2006, pp. 557- 568.

[2] M. Arlitt, D. Krishnamurthy, J. Rolia, "Characterizing the Scalability of a Large Web-based Shopping System", *ACM Trans. on Internet Technology*, v 1, 2001, pp. 44-69.

[3] A. Avritzer, E.J. Weyuker, "Deriving Workloads for Performance Testing", *Software: Practice and Experience*, v 26, 1996, pp. 613 - 633

[4] A. Avritzer, J. Kondek, D. Liu, and E. J. Weyuker, "Software performance testing based on workload characterization," in *Proc. WOSP'2002*, Rome, , pp. 17-24.

[5] R. L. Bagrodia and C.-C. Shen, "MIDAS: Integrated Design and Simulation of Distributed Systems," *IEEE Trans. Software Engineering*, v. 17, 1991, pp. 1042-1058.

[6] S. Balsamo and M. Marzolla. "Simulation Modeling of UML Software Architectures", *Proc. ESM'03,* Nottingham (UK), June 2003

[7] S. Balsamo, A. DiMarco, P. Inverardi, and M. Simeoni, "Model-based Performance Prediction in Software Development", *IEEE Trans. on Software Eng.*, vol. 30, 2004, pp. 295-310.

[8] S. Barber, "Creating Effective Load Models for Performance Testing with Incomplete Empirical Data", in *Proc. 6th IEEE Int. Workshop on Web Site Evolution*, 2004, pp. 51-59.

[9] S. Barber, "Beyond performance testing", parts 1-14, IBM DeveloperWorks, Rational Technical Library, 2004, www-128.ibm.com/developerworks/rational/library/4169.html

[10] S. Barber, "User Community Modeling Language for performance test workloads", http://www-128.ibm.com/ developerworks/rational/library/5219.html, July 2004

[11] S. Barber, "Tester PI: Performance Investigator", *Better Software*, March 2006, pp 20 – 25.

[12] A. Bertolino, R. Mirandola. "Software performance engineering of component-based systems", *Proc.Workshop on Software and Performance, WOSP'2004*, pp 238-242.

[13] T.L. Booth, C.A. Wiecek, "Performance Abstract Data Types as a Tool in Software Performance Analysis and Design", *IEEE Trans. Software Engineering*, v 6, 1980, pp. 138-151.

[14] L. Chung, B. A. Nixon, E. Yu, J. Mylopoulos, *Non-Functional Requirements in Software Engineering*, Kluwer, 2000.

[15] Compuware, *Applied Performance Management Survey*, Oct. 2006.

[16] V. Cortellessa and R. Mirandola, "Deriving a Queueing Network based Performance Model from UML Diagrams", in *Proc. WOSP'2000*, Ottawa, 2000, pp. 58-70.

[17] V. Cortellessa, A. Di Marco, P. Inverardi, "Non-functional Modeling and Validation in Model-Driven Architecture", *Proc 6th Working IEEE/IFIP Conference on Software Architecture* (WICSA 2007), Mumbai, India, 2007.

[18] Czarnecki, K. and U. Eisenecker, *Generative Programming*, Addison Wesley, 2000.

[19] M. Debusmann and K. Geihs, "Efficient and Transparent Instrumentation of Application Components Using an Aspect-Oriented Approach", in *Self-Managing Distributed Systems*, vol. LNCS 2867 Springer, 2003, pp. 209-220.

[20] R. France, B. Rumpe, "Model-driven Development of Complex Systems: A Research Roadmap", in *Future of Software Engineering 2007*, L. Briand and A. Wolf (eds), IEEE-CS Press, 2007.

[21] G. Franks, D.C. Petriu, M. Woodside, J. Xu, P. Tregunno, "Layered Bottlenecks and Their Mitigation", *Proc 3rd Int. Conf. on Quantitative Evaluation of Systems*, Riverside, CA, Sept. 2006.

[22] V. Garousi, L. Briand, Y. Labiche, "Traffic-aware Stress Testing of Distributed Real-Time Systems based on UML Models", *Proc. Int. Conference on Software Engineering*, Shanghai, China, 2006, pp. 391-400.

[23] C. Gill, R. Cytron, and D. Schmidt, "Middleware Scheduling Optimization Techniques for Distributed Real-Time and Embedded Systems", in *Proc 7th Workshop on Object-Oriented Real-time Dependable Systems*, San Diego, Jan. 2002.

[24] C. E. Hrischuk, C. M. Woodside, and J. A. Rolia, "Trace-Based Load Characterization for Generating Software Performance Models", *IEEE Trans. on Software Eng.*, v 25, n 1, Jan 1999, pp. 122-135.

[25] C. E. Hrischuk and C. M. Woodside, "Logical clock requirements for reverse engineering scenarios from a distributed system", *IEEE Trans. Software Eng.*, 28(4), Apr. 2002, 321-339.

[26] IBM, *IBM Rational PurifyPlus, Purify, PureCoverage, and Quantify: Getting Started*, May 2002. G126-5339-00.

[27] T. Israr, M. Woodside, G. Franks, "Interaction Tree Algorithms to Extract Effective Architecture and Layered Performance Models from Traces", *Journal of Systems and Software,* to appear 2007.

[28] X. Jia, D. Li, H. Du, J. Cao, "On Optimal Replication of Data Object at Hierarchical and Transparent Web Proxies", *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 8, 2005, pp. 673 - 685.

[29] M. W. Johnson, *Monitoring and diagnosing application response time with ARM*, in *Proc. IEEE 3rd Int. Workshop on Systems Management*, Newport, RI, USA, Apr.1998, pp. 4-13.

[30] L. Kerber, "Scenario-Based Performance Evaluation of SDL/MSC-Specified Systems", in *Performance Engineering: State of the Art and Current Trends*, LNCS vol. 2047, Springer, 2001.

[31] D. E. Knuth, "An empirical study of FORTRAN programs", *Software Practice and Experience*, vol 1 no 2, Apr. 1971.

[32] J. Kramer, J. Magee, "Self-Managed Systems: An Architectural Challenge", in *Future of Software Engineering 2007*, L. Briand and A. Wolf (eds), IEEE-CS Press, 2007.

[33] D. Krishnamurthy, J.A. Rolia, S. Majumdar, "A Synthetic Workload Generation Technique for Stress Testing Session-Based Systems", *IEEE Trans. on Software Engineering*, v 32, 2006.

[34] M. Litoiu, J.A. Rolia, "Object Allocation for Distributed Applications with Complex Workloads", in *Proc. 11th Int. Conf. Computer Performance Evaluation, Techniques and Tools*, LNCS 1786, 2000, pp. 25-39.

[35] J. W. S. Liu, *Real-Time Systems,* Prentice-Hall, 2000.

[36] Y. Liu, I. Gorton, A. Fekete, "Design-level performance prediction of component-based applications", *IEEE Trans. Software Engineering*, v 31, pp 928 – 941, 2005.

[37] J. P. López-Grao, J. Merseguer, and J. Campos, "From UML Activity Diagrams To Stochastic Petri Nets", in *Proc. WOSP'2004*, Redwood City, CA, 2004, pp. 25-36.

[38] Y. Lu, T. Abdelzaher, C. Lu, L. Sha, X. Liu, "Feedback Control with Queueing-Theoretic Prediction for

Relative Delay Guarantees in Web Servers", in *Proc. 9th IEEE Real-Time and Embedded Technology and Applications Symposium*, Washington, 2003, pp. 208-218.

[39] H. Lucas Jr, "Performance evaluation and monitoring", *ACM Computing Surveys*, 3(3), Sept. 1971, pp 79-91.

[40] M. Lyu: Software Reliability Engineering: A Roadmap, in *Future of Software Engineering 2007*, L. Briand and A. Wolf (eds), IEEE-CS Press, 2007.

[41] P. Maly, C.M. Woodside, "Layered Modeling of Hardware and Software, with Application to a LAN Extension Router", in *Proc 11th Int Conf Performance Evaluation Techniques and Tools*, 2000, pp. 10-24.

[42] A. D. Malony, B. R. Helm, "A theory and architecture for automating performance diagnosis", *Future Generation Computer Systems*, 18(1), Sept. 2001, pp 189-200.

[43] P. Martin, W. Powley, X. Xu, W. Tian, "Automated Configuration of Multiple Buffer Pools", *Computer Journal*, vol. 49, 2006, pp. 487-499.

[44] Stephen J. Mellor, "Agile MDA", Addison Wesley online article, July 23, 2004.

[45] D. Menasce and H. Gomaa, "A Method for Design and Performance Modeling of Client/Server Systems", *IEEE Trans. Software Engineering*, v. 26, 2000, pp. 1066-1085.

[46] D. Mensace, H. Ruan, H. Gommaa, "A Framework for QoS-Aware Software Components", *Proc. WOSP'2004*, Redwood Shores, CA, 2004, pp.186-196.

[47] Merson, P. and Hissam, S. "Predictability by construction" *Posters of OOPSLA 2005*, pp 134-135, San Diego, ACM Press, Oct. 2005.

[48] B.P. Miller, M.D. Callaghan, J.M. Cargille, J.K. Hollingsworth, R.B. Irvin, K.L. Karavanic, K. Kunchithapadam, and T. Newhall, "The Paradyn Parallel Performance Measurement Tool", *IEEE Computer*, v. 28, 1995, pp. 37-46.

[49] A. Mitschele-Theil, B. Muller-Clostermann, "Performance Engineering of SDL/MSC Systems", *Journal on Computer Networks and ISDN Systems*, v. 31, 1999, pp. 1801-1815.

[50] N. Nethercote and J. Seward, "Valgrind: a program supervision framework" *Electronic Notes in Theoretical Computer Science*, 89(2), Oct. 2003, pp. 1-23.

[51] Object Management Group, *MDA Guide*, Version 1.0.1, OMG document omg/2003-06-01, 2003

[52] Object Management Group, *UML Profile for Schedulability, Performance, and Time Specification*, Version 1.1, OMG document formal/05-01-02, Jan 2005.

[53] Object Management Group, *UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE) RFP*, OMG doc. realtime/05-02-06, 2005.

[54] Object Management Group, *Software Process Engineering Metamodel Specification*, formal/05-01-06, 2006.

[55] A.L. Opdahl, "Sensitivity analysis of combined software and hardware performance models: Open queueing networks", *Performance Evaluation*, 22, 1995, pp. 75-92.

[56] D.B. Petriu, C.M. Woodside, "Analysing Software Requirements Specifications for Performance", *Proc. 3rd Int. Workshop on Software and Performance*, Rome, 2002.

[57] Dorin B. Petriu, C.M. Woodside, "An intermediate metamodel with scenarios and resources for generating performance models from UML designs", *Software and Systems Modeling*, vol. 5, no. 4, 2006.

[58] Dorina C. Petriu, H. Shen, "Applying the UML Performance Profile: Graph Grammar based derivation of LQN models from UML specifications", in *Proc. Computer Performance Evaluation - Modelling Techniques and Tools*, LNCS Vol. 2324, Springer, 2002, pp.159-177.

[59] R. H. Reussner, V. Firus, S. Becker, "Parametric Performance Contracts for Software Components and their Compositionality", in *9th Int. Workshop on Component-Oriented Programming*, Oslo, June 2004.

[60] J.A. Rolia, L. Cherkasova, R. Friedrich, "Performance Engineering for EA Systems in Next Generation Data Centres", *Proc. 6th Int. Workshop on Software and Performance*, Buenos Aires, Feb. 2007.

[61] P. C. Roth, B. P. Miller, "On-line Automated Performance Diagnosis on Thousands of Processes", *Proc. ACM Symposium on Principles and Practice of Parallel Programming*, New York, 2006, pp. 69-80

[62] P. K. Sharma, J. Loyall, R. E. Schantz, J. Ye, P. Manghwani, M. Gillen, G. T. Heineman, "Managing End-to-End QoS in Distributed Embedded Applications", *IEEE Internet Computing*, vol. 10, no. 3, 2006, pp. 16-23.

[63] H. A. Sholl and T. L. Booth, "Software Performance Modeling Using Computation Structures", *IEEE Trans on Software Engineering*, v. 1, no. 4 Dec. 1975.

[64] H. Shen, D.C. Petriu, "Performance Analysis of UML Models using Aspect Oriented Modeling Techniques", In *MoDELS 2005* (L. Briand and C. Williams, Eds.), LNCS Vol. 3713, Springer, 2005, pp. 156–170.

[65] C.U. Smith, *Performance Engineering of Software Systems*, Addison Wesley, 1990.

[66] C.U. Smith, L. G. Williams, *Performance Solutions*, Addison-Wesley, 2002.

[67] C.U. Smith, "Software Performance Engineering", *Encyclopedia of Software Engineering,* Wiley, 2002.

[68] C.U. Smith, C. M. Llado. "Performance model interchange format (PMIF 2.0): XML definition and implementation", In *Proc. of 1st Int. Conference on the Quantative Evaluation of Systems (QEST)*, Enschede, The Netherlands, Sept. 2004, pp 38-47.

[69] C.U. Smith, C. M. Lladó, V. Cortellessa, A. diMarco, L. Williams, "From UML models to software performance results: an SPE process based on XML interchange formats", in *Proc WOSP'2005*, Palma de Mallorca, 2005, pp. 87-98.

[70] M. Sopitkamol and D.A. Menasce, "A Method for Evaluating the Impact of Software Configuration Parameters on E-commerce Sites," *Proc. WOSP'2005*, Palma de Mallorca, Spain, 2005, pp.53-64.

[71] C. Szypersky, D. Gruntz, S. Murer, *Component Software: Beyond Object Oriented Programming*, Addison-Wesley, 2002.

[72] A.T. Tai, J.F. Meyer, A. Avizienis., *Software Performability: From Concepts to Applications*, Springer, 1995.

[73] T. Verdickt, B. Dhoedt, F. Gielen, and P. Demeester "Automatic inclusion of Middleware Performance Attributes into Architectural UML Software Models", *IEEE Trans. Software Eng.*, v. 31, 2005, pp.695-711.

[74] K. M. Wasserman, G. Michailidis, and N. Bambos, "Optimal processor allocation to differentiated job flows", *Performance Evaluation*, vol. 63, 2006, pp. 1-14.

[75] E. J. Weyuker and F. I. Vokolos, "Experience with Performance Testing of Software Systems: Issues, an Approach, and Case Study", *IEEE Trans. on Software Engineering*, vol. 26, no. 12 pp. 1147-1156, Dec 2000.

[76] G. Weikum, A. Moenkeberg, C. Hasse, P. Zabback, "Self-tuning database technology and information services: from wishful thinking to viable engineering", *Proc. 28th International Conference on Very Large Data Bases (VLDB 2002)*, pp. 20-31, Hong Kong, CN, 2002.

[77] M. Woodside, "A Three-View Model for Performance Engineering of Concurrent Software", *IEEE Transactions on Software Engineering*, vol. 21, no. 9 pp. 754-767, Sept. 1995.

[78] M. Woodside, C. Hrischuk, B. Selic, and S. Bayarov, "Automated Performance Modeling of Software Generated by a Design Environment", *Performance Evaluation*, vol. 45, no. 2-3 pp. 107-124, 2001.

[79] M. Woodside, D.B. Petriu, K. Siddiqui, "Performance-related Completions for Software Specifications", *Proc 24th Int. Conf. on Software Engineering*, Orlando, 2002.

[80] M. Woodside, D.C. Petriu, D.B. Petriu, H. Shen, T. Israr, J. Merseguer, "Performance by Unified Model Analysis (PUMA)", *Proc. WOSP'2005*, Mallorca, pp 1-12.

[81] C. M. Woodside, T. Zheng, and M. Litoiu, "The Use of Optimal Filters to Track Parameters of Performance Models", in *Proc. 2nd Int. Conf. on Quantitative Evaluation of Systems*, Torino, Italy, 2005, pp. 74-84.

[82] X. Wu and M. Woodside, "Performance Modeling from Software Components", in *Proc.WOSP'2004*, Redwood Shores, Calif., 2004, pp. 290-301.

[83] J. Xu, M. Woodside, and D.C. Petriu, "Performance Analysis of a Software Design using the UML Profile for Schedulability, Performance and Time", in *Proc. 13th Int. Conf. Modeling Techniques and Tools for Computer Performance Evaluation*, Urbana, USA, Sept. 2003

[84] C. Yilmaz, A. Krishna, A. Memon, A. Porter, D. C. Schmidt, A. Gokhale, and B. Natarajan, "Main Effects Screening: A Distributed Continuous Quality Assurance Process for Monitoring Performance Degradation in Evolving Software Systems", in *Proc. 27th International Conference on Software Engineering (ICSE '05)*, St. Louis, MO, May 2005.

[85] T. Zheng, M. Woodside, "Heuristic Optimization of Scheduling and Allocation for Distributed Systems with Soft Deadlines", in *Proc. 13th Int. Conf. Modelling Techniques and Tools for Computer Performance Evaluation*, Urbana, USA, 2003.