Model-Based Performance Prediction in Software Development: A Survey

Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, *Member*, *IEEE*, and Marta Simeoni

Abstract—Over the last decade, a lot of research has been directed toward integrating performance analysis into the software development process. Traditional software development methods focus on software correctness, introducing performance issues later in the development process. This approach does not take into account the fact that performance problems may require considerable changes in design, for example, at the software architecture level, or even worse at the requirement analysis level. Several approaches were proposed in order to address early software performance analysis. Although some of them have been successfully applied, we are still far from seeing performance analysis integrated into ordinary software development. In this paper, we present a comprehensive review of recent research in the field of model-based performance prediction at software development time in order to assess the maturity of the field and point out promising research directions.

Index Terms—Software verification, performance modeling and prediction, integrated environments.

1 Introduction

Over the last decade, research has addressed the importance of integrating quantitative validation in the software development process, in order to meet nonfunctional requirements. Among these, performance is one of the most influential factors to be considered. Performance problems may be so severe that they can require considerable changes in design, for example, at the software architecture level. In the worse cases, they can even impact the requirements level.

Traditional software development methods focus on software correctness, introducing performance issues later in the development process. This style of developing has been often referred to as a "fix-it-later" approach.

In the research community, there has been a growing interest in the subject and several approaches to early software performance predictive analysis have been proposed. In this paper, we are interested in approaches which fall into the category that proposes the use of performance models to characterize the quantitative behavior of software systems. Although several of these approaches have been successfully applied, we are still far from seeing performance prediction integrated into ordinary software development.

The goal of this work is to present a comprehensive review of recent research in the field of model-based performance prediction at software development time in order to assess the maturity of the field and point out promising research directions.

In this paper, we mean by *software performance* the process of predicting (at early phases of the life cycle) and evaluating (at the end), based on performance models, whether the software system satisfies the user performance goals.

This definition outlines two basic features of the methods we review: the existence of a performance model suitably coupled with the system software artifacts and the evaluation of the performance model as the means to obtain software performance results. In our opinion, these are necessary conditions for a method to aim at a seamless integration in ordinary software development environments since they both exhibit a good degree of automation.

Approaches in the area of runtime monitoring of system performance are out of the scope of the present paper since they imply the availability of system implementation. The paper concerns the methodological issues of suitably integrating performance prediction in the early phases of the software life cycle. Moreover, since we are focusing on the evaluation of performance models to carry out quantitative validation, we do not consider measurement-based techniques that apply to the various software artifacts [12], [18].

The review we carry out analyzes the approaches with respect to a set of relevant dimensions: software specification, performance model, evaluation methods, and level of automated support for performance prediction.

From the software point of view, the software performance predictive process is based on the availability of software artifacts that describe suitable abstraction of the final software system. Requirements, software architectures, specifications, and design documents are examples of artifacts. Since performance is a runtime attribute of a software system, performance analysis requires suitable descriptions of the software runtime behavior, from now on referred to as dynamics. For example, finite state automata

- *S. Balsamo and M. Simeoni are with the Dipartimento di Informatica, Universitá di Venezia, Via Torino 155, 30173 Mestre, Venezia. E-mail: {balsamo, simeoni}@dsi.unive.it.*
- *A. Di Marco and P. Inverardi are with the Dipartimento di Informatica, Universitá di L'Aquila, Via Vetoio 1, 67010 Coppito, L'Aquila. E-mail: {adimarco, inverard}@di.univaq.it.*

Manuscript received 18 Dec. 2002; revised 4 Mar. 2004; accepted 4 Mar. 2004.
Recommended for acceptance by B. Cheng.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 118003.

and message sequence charts are largely used behavioral models. As far as performance analysis is concerned, we concentrate on model-based approaches that can be applied to any phase of the software life cycle to obtain figures of merit characterizing the quantitative behavior of the system. The most used performance models are queueing networks, stochastic Petri nets, stochastic process algebra, and simulation models.

Analytical methods and simulation techniques can be used to evaluate performance models in order to get performance indices. These can be classical resource-oriented indices such as throughput, utilization, response time, and/or new figures of merit such as power consumption related to innovative software systems, e.g., mobile applications.

A key factor in the successful application of early performance analysis is automation. This means the availability of tools and automated support for performance prediction in the software life cycle. Although complete integrated proposals to software development and performance prediction are not yet available, several approaches provide automation of portions of it. From software specification to performance modeling to evaluation, methods and tools have been proposed to partially automate this integrated process.

The contribution of this paper is to provide a critical reading of the most relevant approaches in software performance prediction with respect to their suitability to be smoothly integrated into the software life cycle. The reviewed methods are classified and discussed along three dimensions which synthesize a set of significant features. Based on this classification, we propose our understanding of the future research directions in the area.

The paper is structured as follows: Section 2 details the problem domain. Section 3 carries out a comprehensive review of the integrated methods to model-based software performance prediction. In Section 4, we discuss the various methodologies and summarize the state of the art identifying challenging future research. The last section presents concluding remarks. In the full version of this paper [15], the interested reader can find in the Appendix a summary of the most commonly used formalisms to specify software dynamics, performance models, and evaluation methods. For the sake of clarity, we report in Table 1 the list of abbreviations used throughout the paper.

2 Setting the Context

Software performance aims at integrating performance analysis in the software domain. Historically, the approach was to export performance modeling and measurements from the hardware domain to software systems. This was rather straightforward when considering the operating system domain, but it assumed a new dimension when the focus was directed toward software applications. Moreover, with the increase of software complexity, it was recognized that software performance could not be faced locally at the code level by using optimization techniques since performance problems often result from early design choices. This awareness pushed the need to anticipate performance analysis at earlier stages in the software development [68], [52].

In this context, several approaches have been proposed [75], [76], [77]. In the following, we focus on the integration of performance analysis at the earliest stages of the software life cycle, namely, software design, software architecture, and software specification. We will review the most important approaches in the general perspective of how each one integrates into the software life cycle.

To carry out our review, we consider a generic model of software life cycle as presented in Fig. 1.

We identify the most relevant phases in the software life cycle. Since the approaches we are reviewing aim at addressing performance issues early on in the software development cycle, our model is detailed with respect to the requirement and architectural phases. Moreover, since performance is a system runtime attribute, we will focus on dynamic aspects of the software models used in the different phases.

The various approaches differ with respect to several dimensions. In particular, we consider the software dynamics model, the performance model, the phase of the software development in which the analysis is carried out, the level of detail of the additional information needed for the analysis, and the software architecture features of the system under analysis, e.g., specific architectural patterns such as client-server, and others. All these dimensions are then synthesized through the three following indicators: the *integration level of the software model with the performance model*, the *level of integration of performance analysis in the software life cycle*, and the *methodology automation degree*, as shown in Fig. 2. We use them to classify the methodologies as reported in Section 4. The level of integration of the software and of the performance models ranges from syntactically related models to semantically related models to a unique comprehensive model. This integration level qualifies the mapping that the methodologies define to relate the software design artifacts with the performance model. A high level of integration means that the performance model has a strong semantic correspondence with the software model. Syntactically related models permit the definition of a syntax driven translation from the syntactic specification of software artifacts to the performance model. The unique comprehensive model allows the integration of behavioral and performance analysis in the same conceptual framework. With the level of integration of performance analysis in the software life cycle, we identify the precise phase at which the analysis can be carried out. In the context we are dealing with, the earlier the prediction process can be applied the better the integration with the development process is obtained. For each methodology, we also single out the software development phases in which the specific information required to perform the analysis is collected, received, or supplied. This information is synthesized in Fig. 4, discussed in Section 4, and it is used to explicitly state the requirements that each methodology puts to software designers in terms of additional information required for performance analysis. It is worth noting that these requirements can be very demanding, thus restricting the general applicability of the methodology. We associate each required information to

TABLE 1
List of Abbreviations

Abbreviation	Meaning
ADL	Architectural Description Language
AMG	Analysis Model Generator
CASE tool	Computer Aided Software Engineering tool
CLISSPE	CLient/Server Software Performance Evaluation
EG	Execution Graph
EMPA	Extended Markovian Process Algebra
EQN	Extended Queueing Network
GSPN	Generalized Stochastic Petri Net
ITU	International Telecommunication Union
LTS	Label Transition System
LQN	Layered Queueing Network
MSC	Message Sequence Chart
OAT	Object-oriented performance modeling and Analysis Tool
OMG	Object Management Group
OMT	Object Modeling Technique
PA	Process Algebra
PAMB	Performance Analysis Model Builder
PASA	Performance Assessment of Software Architectures
PEPA	Performance Evaluation Process Algebra
PN	Petri Net
QN	Queueing Network
SA	Software Architecture
SMG	Simulation Model Generator
SimML	Simulation Modeling Language
SPA	Stochastic Process Algebra
SPE	Software Performance Engineering
SPN	Stochastic Petri Net
STPN	Stochastic Timed Petri Net
TIPP	TIme Process and Performability evaluation
UCM	Use Case Map
UML	Unified Modeling Language
XML	eXtensible Markup Language
XMI	XML Metadata Interchange
XSLT	eXtensible Stylesheet Language Transformations

the phase of the software life cycle where it would be naturally available. There exist methods that assume the availability of this information at earlier phases of the software life cycle in order to carry out the predictive analysis. In this case, their underlying assumption is that this information is available somehow, for example, through an operational profile extracted from similar systems or by assuming hypothetical implementation scenarios. We will mention this kind of assumption for each presented methodology. The last dimension refers to the degree of automation that the various approaches can support. It indicates the potentiality of automation and characterizes the maturity of the approach and the generality of its applicability.

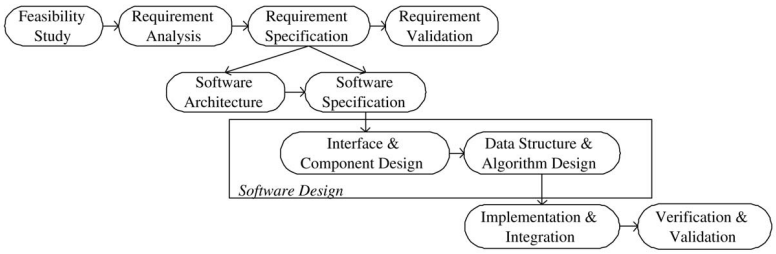

Fig. 1. Generic software life cycle model.

The classification schema illustrated in Fig. 2 is inspired by Enslow's model of distribution [27].

The ideal methodologies fall at the bottom of the two rightmost columns, since they should have high integration of the software model with the performance model, high level of integration of performance analysis with the life cycle, i.e., from the very beginning, and high degree of automation.

3 Integrated Methods for Software Performance Engineering

In this section, we review approaches that propose a general methodology for software performance focusing on early predictive analysis. They refer to different specification languages and performance models, and consider different tools and environments for system performance evaluation. Besides these, other proposals in the literature present ideas on model transformation through examples or case studies (e.g., [38], [49], [60], [59]). Although interesting, these approaches are still preliminary, thus we will not treat them in the following comparison. Background material concerning notations to specify software dynamics and performance models is summarized in [15].

The various methodologies are grouped together and discussed on the basis of the type of underlying performance model. In the same logical grouping, approaches are discussed chronologically. As introduced in Section 2, our analysis refers to the general model of software life cycle presented in Fig. 1 and outlines the three classification dimensions shown in Fig. 2. For each methodology, we describe the software development phases in which it collects, receives, or supplies the information required for performance analysis. Such information is available in Fig. 4, which is presented and commented in Section 4.

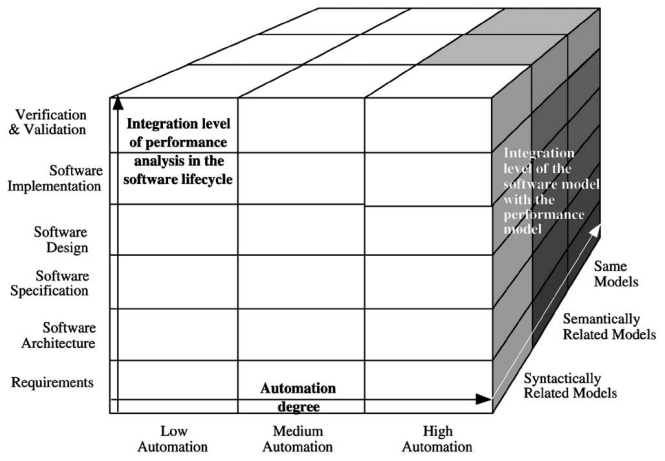

Fig. 2. Classification dimensions of software performance approaches.

3.1 Queueing Network-Based Methodologies

In this section, we consider a set of methodologies which propose transformation techniques to derive Queueing Network (QN) based models—possibly Extended QN (EQN) or Layered QN (LQN) [46], [47], [67], [72], [74], [28]—from Software Architecture (SA) specifications. Some of the proposed methods are based on the Software Performance Engineering (SPE) methodology introduced by Smith in her pioneer work [68].

3.1.1 Methodologies Based on the SPE Approach

The SPE methodology [68], [70] was the first comprehensive approach to the integration of performance analysis into the software development process, from the earliest stages to the end. It uses two models: the software execution model and the system execution model. The first takes the form of Execution Graphs (EG) that represent the software execution behavior; the second is based on QN models and represents the system platform, including hardware and software components. The analysis of the software model gives information about the resource requirements of the software system. The obtained results, together with information about the hardware devices, are the input parameters of the system execution model, which represents the model of the whole software/hardware system.

M1. The first approach based on the SPE methodology was proposed by Williams and Smith in [78], [70]. They apply the SPE methodology to evaluate the performance characteristics of a software architecture specified by using the Unified Modeling Language (UML) diagrams [80], that is, Class and Deployment diagrams and Sequence Diagrams enriched with ITU Message Sequence Chart (MSC) [44] features. The emphasis is in the construction and analysis of the software execution model, which is considered the target model of the specified SA and is obtained from the Sequence Diagrams. The Class and Deployment diagrams contribute to complete the description of the SA, but are not involved in the transformation process. This approach was initially proposed in [71], which describes a case study and makes use of the tool SPE•ED for performance evaluation. In [79], the approach is embedded into a general method called PASA (Performance Assessment of Software Architectures) which aims at giving guidelines and methods to

determine whether an SA can meet the required performance objectives.

SPE•ED is a performance modeling tool specifically designed to support the SPE methodology. Users identify the key scenarios, describe their processing steps by means of EG, and specify the number of software resource requests for each step. A performance specialist provides overhead specifications, namely, the computer service requirements (e.g., CPU, I/O) for the software resource requests. SPE•ED automatically combines the software models and generates a QN model, which can be solved by using a combination of analytical and simulation model solutions. SPE•ED evaluates the end-to-end response time, the elapsed time for each processing step, the device utilization, and the time spent at each computer device for each processing step.

M2. An extension of the previous approach was developed by Cortellessa and Mirandola in [23]. The proposed methodology, called PRIMA-UML, makes use of information from different UML diagrams to incrementally generate a performance model representing the specified system. This model generation technique was initially proposed in [22]. The technique considered OMT-based object-oriented specification of systems (Class diagrams, Interaction diagrams and State Transition diagrams) and defines an intermediate model, called Actor-Event Graph, between the specification and the performance model.

In PRIMA-UML, SA are specified by using Deployment, Sequence, and Use Case diagrams. The software execution model is derived from the Use Case and Sequence diagrams, and the system execution model from the Deployment diagram. Moreover, the Deployment diagram allows for the tailoring of the software model with respect to information concerning the overhead delay due to the communication between software components. Both Use Case and Deployment diagrams are enriched with performance annotations concerning workload distribution and parameters of hardware devices, respectively.

M3. In [34], the PRIMA-UML methodology was extended in order to cope with the case of mobile SA by enhancing its UML description to model the mobility-based paradigms. The approach generates the corresponding software and system execution models allowing the designer to evaluate the convenience of introducing logical mobility with respect to communication and computation costs. The authors define extensions of EG and EQN to model the uncertainty about the possible adoption of code mobility.

M4. In [20], Cortellessa et al. focus on the derivation of a LQN model from an SA specified by means of a Class diagram and a set of Sequence diagrams, generated by using standard CASE tools. The approach clearly identifies all the supplementary information needed by the method to carry out the LQN derivation. These include platform data, e.g., configuration, resource capacity, and the operational profile which includes the user workload data. Moreover, in case of distributed software, other input documents are the software module architecture, the client/server structure, and the module-platform mapping.

The method generates intermediate models in order to produce a global precedence graph which identifies the execution flow and the interconnections among system components. This graph, together with the workload data, the software module architecture, and the client/server structure, are used to derive the extended EG. The target LQN model is generated from the extended EG and the resource capacity data.

Other SPE-based approaches have been proposed by Petriu and coauthors. Since they are also based on patterns, we describe them in the next section.

3.1.2 Architectural Pattern-Based Methodologies

The approaches described hereafter consider specific classes of systems, identified by architectural patterns in order to derive their corresponding performance models. Architectural patterns characterize frequently used architectural solutions. Each pattern is described by its structure (what are the components) and its behavior (how they interact).

A first approach based on architectural patterns for client/server systems is presented in [32], [33], where Gomaa and Menascé investigate the design and performance modeling of component interconnection patterns, which define and encapsulate the way client and server components of SA communicate with each other via connectors. The authors, rather than proposing a transformational methodology, describe the pattern through Class and Collaboration diagrams and directly show their corresponding EQN models. It is worth mentioning that the approach has been the first to deal with component-based SA.

M5. In [64], [62], [35], Petriu et al. propose three conceptually similar approaches where SA are described by means of architectural patterns (such as pipe and filters, client/server, broker, layers, critical section, and master-slave) whose structure is specified by UML Collaboration diagrams and whose behavior is described by Sequence or Activity diagrams.

The three approaches follow the SPE methodology and propose systematic methods of building LQN models of complex SA based on combinations of the considered patterns. Such model transformation methods are based on graph transformation techniques. We now discuss the details of the various approaches.

- In [64], SA are specified by using UML Collaboration, Sequence, Deployment, and Use Case diagrams. Sequence diagrams are used to obtain the software execution model represented as a UML Activity diagram. The UML Collaboration diagrams are used to obtain the system execution model, i.e., an LQN model. Use Case diagrams provide information on the workloads and Deployment diagrams allow for the allocation of software components to hardware sites. The approach generates the software and system execution models by applying graph transformation techniques, automatically performed by a general-purpose graph rewriting tool.
- In [62], the authors extend the previous work by using the UML performance profile [81] to add performance annotations to the input models, and by accepting UML models expressed in XML

notation as input. Moreover, the general-purpose graph rewriting tool for the automatic construction of the LQN model, has been substituted by an ad-hoc graph transformation implemented in Java.

- The third approach proposed in [35] uses the eXtensible Stylesheet Language Transformations (XSLT), to carry out the graph transformation step. XSLT is a language for transforming a source document expressed in a tree format (which usually represents the information in a XML file) into a target document expressed in a tree format. The input contains UML models in XML format, according to the standard XML Metadata Interchange (XMI) [82], and the output is a tree representing the corresponding LQN model. The resulting LQN model can be in turn analyzed by existing LQN solvers after an appropriate translation into textual format.

M6. Menascé and Gomaa presented in [55], [56] an approach to the design and performance analysis of client/server systems. It is based on CLISSPE (CLIent/Server Software Performance Evaluation), a language for the software performance engineering of client/server applications [54]. A CLISSPE specification is composed of a *declaration section* containing clients and client types, servers and server types, database tables and other similar information, a *mapping section* allocating clients and servers to networks, assigning transactions to clients, etc., and a *transaction specification section* describing the system behavior.

The CLISSPE system provides a compiler which generates the QN model, estimates some model parameters, and provides a performance model solver. For the specification of a client/server system the methodology uses UML Use Case diagrams to specify the functional requirements, Class diagrams for the structural model, and Collaboration diagrams for the behavioral model. From these diagrams, the methodology derives a CLISSPE specification. A relational database is automatically derived from the Class diagram. The CLISSPE transaction specification section is derived (not yet automatically) from Use Case diagrams and collaboration diagrams, and references the relational database derived from the structural model. Moreover, the CLISSPE system uses information on the database accesses (of the transaction specification section) to automatically compute service demands and estimate CPU and I/O costs.

In order to complete the CLISSPE declaration and mapping sections, the methodology requires the specification of the client/server architecture with the related software/hardware mapping annotated with the performance characteristics such as processor speeds, router latencies, etc.

3.1.3 Methodologies Based on Trace-Analysis

In this section, we consider approaches based on the generation and analysis of traces (sequence of events/actions) from dynamic description of the software system.

M7. The first approach we consider proposes an algorithm that automatically generates QN models from software architecture specifications described by means of MSC [1] or by means of Labeled Transition Systems (LTS) [4]. A key point of this approach is the assumption that the SA dynamics, described by MSC or LTS, is the only available system knowledge. This specification models the interactions among components. This allows the methodology to be applied in situations where information concerning the system implementation or deployment are not yet available.

The approach analyzes the SA dynamic specification in terms of the execution traces (sequences of events exchanged between components) it defines, in order to single out the real degree of parallelism among components and their dynamic dependencies. Only the components that actually can behave concurrently will correspond to service centers of the QN model of the SA description. In the QN model, the dynamic activation of software components and connectors is modeled by customers' service time. Concurrent component activation of software components is represented by customers' concurrent activity that possibly compete for the use of shared resources. Synchronous communication of concurrent software components are modeled by service centers with finite or zero capacity queues and an appropriate blocking protocol. The analysis of the QN model of the SA leads to the evaluation of a set of performance indices, like throughput and mean response time, that are then interpreted at the SA level. The model parameter instantiations correspond to potential implementation scenarios. Performance results are used to provide insights on how to carry out refinements to the SA design.

M8. In [73], Woodside et al. describe a methodology to automatically derive an LQN model from a commercial software design environment called ObjecTime Developer [58] by means of an intermediate prototype tool called PAMB (Performance Analysis Model Builder). The application domain of the methodology is real-time interactive software and it encompasses the whole development cycle, from the design stage to the final product.

ObjecTime Developer allows the designer to describe a set of communicating actor processes, each controlled by a state machine, plus data objects and protocols for communications. It is possible to "execute" the design over a scenario by inserting events, stepping through the state machines, and executing the defined actions. Moreover, the tool can generate code from the system design. The approach in [73] takes advantage of such code generation and scenario execution capabilities for model-building. The prototype tool PAMB, integrated with ObjecTime Developer, keeps track of the execution traces and captures the resource demands obtained by executing the generated code in different execution platforms. Essentially, the trace analysis allows the building of the various LQN submodels (one for each scenario) which are then merged into a global model, while the resource demand data provide the model parameters. After solving the model through an associated model solver, the PAMB environment reports the performance results by means of performance annotated MSC and graphs of predictions.

M9. More recently, Woodside et al. present in [63] an approach to performance analysis from requirements to architectural phases of the software life cycle. This approach derives LQN performance models from system scenarios described by means of Use Case Maps (UCM) [10].

The UCM specification is enriched with performance annotation. The approach defines where and how the diagrams have to be annotated and the default values to be used when performance data are missing.

The derivation of LQN models from annotated UCM is quite direct, due to the close correspondence between UCM and LQN basic elements. The derivation is defined on a path by path basis, starting from UCM start points. The identification of the component interaction types, however, is quite complex since the UCM notation does not allow the specification of synchronous, asynchronous, and forwarding communication mechanisms. The approach describes an algorithm which derives such information from the UCM paths, by maintaining the unresolved message history while traversing a UCM path.

The UCM2LQN tool automatizes the methodology and it has been integrated into a general framework called UCM Navigator. This allows the creation and editing of UCM, supports scenario definitions, generates LQN models, and exports UCM specifications as XML files.

3.1.4 UML for Performance

In this section, we consider efforts that have been pursued entirely in the UML framework in order to make performance analysis possible by starting from UML software descriptions.

The UML Profile for Scheduling, Performance, and Time was described in [81] and has been adopted as an official OMG standard in March 2002. In general, a UML profile defines a domain-specific interpretation of UML; it might be viewed as a package of specializations of general UML concepts that capture domain-specific variations and usage patterns. Additional semantic constraints introduced by the UML profile must conform to the standard UML semantics. To specify a profile, UML extensibility mechanisms (i.e., stereotypes, tagged values, constraints) are used.

The main aims of the UML Profile for Scheduling, Performance, and Time (Real-time UML standard) are to identify the requirements for enabling performance and scheduling analysis of UML models. It defines standard methods to model physical time, timing specifications, timing services and logical and physical resources, concurrency and scheduling, software and hardware infrastructure, and their mapping. Hence, it provides the ability to specify quantitative information directly in UML models allowing quantitative analysis and predictive modeling. This profile has been defined to facilitate the use of analysis methods and to automate the generation of analysis models and of the analysis process itself. Analysis methods considered in the profile are scheduling analysis and performance analysis based on queueing theory.

M10. The approach introduced by Kähkipuro in [48] is quite different from the others described in this section. The proposed framework consists of three different performance model representations and of the mappings among them. The starting representation is based on UML. The key point of this approach is the use of UML as a new way to represent performance models. This approach proposes a UML-based performance modeling notation (i.e., a notation compatible with the UML design description) which can be used in the UML specification of a system, in order to specify performance elements besides the pure functional ones. The UML representation is then automatically mapped into a textual representation, which retains only the performance aspects of the system, and it is further translated into an extended QN model representing the simultaneous resource possessions, synchronous resource invocations, and recursive accesses to resources. Such a model can be solved by using approximate or simulation techniques, and the results can be translated back to the textual representation and the UML diagrams, thus realizing a feedback mechanism. The approach has been partially implemented in a prototype tool called OAT (Object-oriented performance modeling and Analysis Tool).

Note that this approach does not really propose a transformation methodology from an SA specification to a performance model since the three steps (or representations) of the framework just give three equivalent views of the modeled system with respect to performance. In this approach, a designer must have performance skills, besides UML knowledge, to produce a correct diagram. However, the approach lifts up the transformation step to the specification level. In fact, in order to obtain a real model transformation methodology, it would be sufficient to add a further level on top of the whole framework in order to produce extended UML diagrams annotated with performance information out of purely functional oriented UML diagrams.

3.2 Process-Algebra-Based Approaches

M11. Several Stochastic extensions of Process Algebras [39], [40] (SPA) have been proposed in order to describe and analyze both functional and performance properties of software specifications within the same framework. Among these, we consider TIPP (TIme Processes and Performability evaluation) [41], EMPA (Extended Markovian Process Algebra) [17], [11], and PEPA (Performance Evaluation Process Algebra) [37], [30], which are all supported by appropriate tools (the TIPP tool, PEPA Workbench, and Two Towers for EMPA).

All of them associate exponentially distributed random variables to actions, and provide the generation of a Markov chain out of the semantic model (LTS enriched with time information) of a system. Beside exponential actions, passive and immediate actions are also considered. The main differences among PEPA, EMPA and TIPP concern the definition of the rate of a joint activity which arises when two components cooperate or synchronize, and the use of immediate actions. Different choices on these basic issues induce differences to the expressive power of the resulting language. We refer to [40] for an accurate discussion about this topic.

The advantage of using PEPA, EMPA, or TIPP for software performance is that they allow the integration of functional and nonfunctional aspects and provide a unique reference model for software specification and performance. However, from the performance evaluation

viewpoint, the analysis usually refers to the numerical solution of the underlying Markov chain which can easily lead to numerical problems due to the state space explosion. On the software side, the software designer is required to be able to specify the software system using process algebras and to associate the appropriate performance parameters (i.e., activity rates) to actions.

In order to overcome this last drawback, Pooley describes in [59] some preliminary ideas on the derivation of SPA models from UML diagrams. The starting point is the specification of an SA by means of a combined diagram consisting of a Collaboration diagram with embedded Statecharts of all the collaborating objects. The idea is then to produce an SPA description out of each Statechart and then to combine the obtained descriptions into a unique model.

Balsamo et al. introduced in [9] an SPA-based Architectural Description Language (ADL) called Æmilia (an earlier version called ÆMPA can be found in [11]), whose semantics is given in terms of EMPA specifications. Æmilia aims at facilitating the designer in the process algebra-based specification of software architectures, by means of syntactic constructs for the description of architectural components and connections. Æmilia is also equipped with checks for the detection of possible architectural mismatches. Moreover, for Æmilia specifications, a translation into QN models has been proposed in order to take advantage of the orthogonal strengths of the two formalisms: formal techniques for the verification of functional properties for Æmilia (SPA in general) and efficient performance analysis for QN.

3.3 Petri-Net-Based Approaches

M12. Like SPA, Stochastic Petri Nets (SPN) are usually proposed as a unifying formal specification framework, allowing the analysis of both functional and nonfunctional properties of systems. There are a lot of Stochastic Petri Net frameworks (e.g., GreatSPN, HiQPN, DSPNExpress 2000, and many others which can be found in [61]) allowing the specification and the functional/quantitative analysis of a Petri Net model.

Recently, approaches to the integration of UML specifications and Petri Nets have been proposed [49], [16]. In [49], the authors present some ideas on the derivation of a General Stochastic Petri Net (GSPN) [3], [7] from Use Cases diagrams and combined Collaboration and Statecharts diagrams. The idea is to translate the Statechart associated with each object of the Collaboration diagram into a GSPN (where states and transitions of the Statechart become places and transitions in the net, respectively) and then to combine the various nets into a unique model.

In [16], the authors propose a systematic translation of Statecharts and Sequence Diagrams into GSPN. The approach consists of translating the two type of diagrams into two separate labeled GSPN.

The translation of a Statechart gives rise to one labeled GSPN per unit where a unit is a state with all its outgoing transitions. The resulting nets are then composed over places with equal labels in order to obtain a complete model. Similarly, the translation of a Sequence diagram consists of modeling each message with a labeled GSPN subsystem and then composing such subsystems by taking into account the causal relationship between messages belonging to the same interaction and defining the initial marking of the resulting net. The final model is obtained by building a GSPN model by means of two composing techniques.

3.4 Methodologies Based on Simulation Methods

We shall now consider two approaches based on simulation models [13]. They use simulation packages in order to define a simulation model whose structure and input parameters are derived from UML diagrams.

M13. The first approach, proposed by de Miguel et al. in [26], focuses on real time systems, and proposes extensions to UML diagrams to express temporal requirements and resource usage. The extension is based on the use of stereotypes, tagged values, and stereotyped constraints. SA are specified using the extended UML diagrams without restrictions on the type of diagrams to be used. Such diagrams are then used as input for the automatic generation of the corresponding scheduling and simulation models via the Analysis Model Generator (AMG) and Simulation Model Generator (SMG), respectively. In particular, SMG generates OPNET models [57], by first generating one submodel for each application element and then combining the obtained submodels into a unique simulation model. The approach provides a feedback mechanism: after the model has been analyzed and simulated, some results are included into the tagged values of the original UML diagrams. This is a relevant feature, which helps the SA designer in interpreting the feedback from the performance evaluation results.

M14. The second approach, proposed by Arief and Speirs in [5], presents a simulation framework named Simulation Modeling Language (SimML) to automatically generate a simulation Java program (by means of the JavaSim tool [45]) from the UML specification of a system that realizes a process oriented simulation model. SimML allows the user to draw Class and Sequence diagrams and to specify the information needed for the automatic generation of the simulation model. The approach proposes an XML translation of the UML models, in order to store the information about the design and the simulation data in a structured way.

3.5 A Methodology Based on Stochastic Processes

All the approaches presented in this section so far consider QN, SPA, SPN, or simulation-based performance models. In this section, we describe an approach which considers generalized semi-Markov processes, i.e., stochastic processes where general distributions (and not only memoryless ones) are allowed and the Markovian property is only partially fulfilled.

M15. The approach in [51] proposes a direct and automatic translation of system specifications given by UML State diagrams or Activity diagrams, into a corresponding discrete-event stochastic system, namely, a generalized semi-Markov process. The first step of the approach consists of introducing extensions to UML State diagrams and Activity diagrams to associate events with exponentially distributed durations and deterministic delays. The enhanced UML diagrams are then mapped onto a generalized semi-Markov process by using an efficient algorithm for the state space generation.

The approach has been implemented using the tool DSPNexpress 2000 (information in [61]) which supports the quantitative analysis of discrete-event stochastic processes. DSPNexpress imports UML diagrams from some commercial UML design package, and adds the timing information by using a graphical user interface.

The approach has two main drawbacks, namely, the use of nonstandard UML notation for the additional timing information and the possible state space explosion in the generation of the state transition graph out of the UML diagrams.

4 Comparison and Classification

In this section, we provide a synthesis of the methodologies previously surveyed in light of the several features outlined in Section 2. The table in Fig. 3 summarizes all the methodologies and their characteristics. Each row refers to a methodology. The first column indicates the methodology label that will be used in Fig. 5, while the second column refers to the section of Section 3 presenting the methodology. The third and fourth columns indicate the behavioral and performance models, respectively. The former is the starting point of the methodology and the latter represents the target model for the analysis. The fifth and sixth columns indicate potential constraints implied by the methodology. These can be related to the software system architecture or to a specific application domain, e.g., real time systems. The life cycle phase column reports the first software development phase in which the methodology can be applied. The eighth, ninth, and tenth columns are quality attributes that we derive from the study of the various methodologies. Information level represents the amount of information that has to be provided in order to make the methodology applicable. This encompasses information like operational profile, resource workloads, as detailed in Fig. 4. Feedback indicates that a methodology has explicitly addressed the issue of providing some sort of feedback from performance analysis to the software designer. This means, for example, that, from a bad throughput figure, it is easy to single out which are the software components and/or interactions responsible. Automation degree refers to the suitability for automation of a methodology. It is worth noting that this does not indicate the current achieved automation, which is instead considered in the last column. The tool column mentions the tools that support the automated portion of the methodology, if any.

Note that there is a dependency between the information level and the life cycle phase. The more information needed, the later in the software development cycle the analysis is performed. This means that some methodologies, although starting from abstract software descriptions, are applicable only from the design level onward, as far as the life cycle phase is concerned. The kind of additional performance information required by each methodology can be found in the table of Fig. 4. Note that one could also imagine that this information does somehow exist, e.g., can be predicted by using execution scenarios. However, for classification purposes, we consider the development phase in which this kind of information would naturally be available and relate the methodology to that phase. More precisely, the table in Fig. 4 depicts the various phases of the software life cycle in the columns and the considered methodologies in the rows. Each row is relative to one methodology and reports the information needed for performance analysis purposes, with respect to the significant phases of the software life cycle.

Looking at Fig. 3, we can make the following observations.

- Most of the methodologies make use of UML or UML-like formalisms to describe behavioral models. This indicates a general tendency which is driven by the need to integrate with standard practice development environments.
- QN are the preferred performance models. This depends on two factors: the abstraction level of the QN formalism which makes it suitable to describe software architecture models, and high-level software design and the availability of efficient algorithms and tools to evaluate the model. Moreover, QN models can be extended to better reflect software characteristics like communication patterns and hierarchical structure (e.g., EQN, LQN).
- Architectural and application domain constraints represent a serious attempt to make an integrated approach to performance prediction analysis efficient and applicable. Identifying specific application domains and architectural structures can be the means for achieving scalability and/or modularity in performance analysis. This research area is even more relevant considering the development of component-based systems. In this setting, components are plugged into fixed underlying architectural structures in order to produce different systems.
- As far as the additional information required for performance analysis is concerned, Fig. 4 summarizes the kind of information needed by each approach. Performance requirements are obviously always assumed to exist. All the methods explicitly based on SPE, see Section 3.1.1, assume to have available performance related information at each phase of the software life cycle, until the detailed design is obtained. This obviously prevents their use at early stages. The same consideration applies to **M5**, which although based on architectural patterns, follows the SPE approach and requires detailed information such as resource usage and loop repetitions that can only be available at detailed design level. Differently, **M6**, also working with architectural patterns, requires information on the hardware platform that in many cases can be either available at requirement phase or supplied as possible alternative platform scenarios.

  The three methods based on trace analysis, **M7**, **M8**, and **M9**, see Section 3.1.3, are less demanding in terms of actual information, and two of them, **M7** and **M9**, can be applied at software architecture and requirement level, respectively.

  M10, as discussed in Section 3.1.4, suggests a UML-based design to achieve performance modeling. Therefore, all the performance related information is integrated into design artifacts.

Approach	Reference in Section 3	Behavioral Model	Performance Model	Architectural Constraint	Application Domain	Life cycle Phase	Information Level	Feedback	Automation Degree	Tool
M1	3.1.1	Annotated MSC	QNM	—	General	Design	High	No	High	SPE•ED
M2	3.1.1	UML Sequence	EQNM	—	General	Design	High	No	High	—
M3	3.1.1	UML Diagrams	EQNM	—	Mobility Code	Design	High	No	High	—
M4	3.1.1	UML Sequence	LQN	Client/Server	General	Design	High	No	Medium/ Low	—
M5	3.1.2	UML Activity Collaboration	LQN	Architectural Patterns	General	Design	High	No	High	Implementations (not available)
M6	3.1.2	UML Collaboration	QNM	Client/Server	General	Design	High	No	Medium	CLISSPE System (not available)
M7	3.1.3	Annotated MSC	QNM	—	General	Architecture Design	Low	No	High	—
M8	3.1.3	ROOM Notation: State Machine	LQN	—	Real-Time Interactive	Design	High	Yes	High	PAMB
M9	3.1.3	UCM	LQN	—	General	Reqs and Architectural Design	High	No	High	UCM2LQN UCM Navigator
M10	3.1.4	UML Diagrams	QN (closed multiclass product form)	—	General	Design	High	Yes	High	OAT
M11	3.2	Process Algebra	SPA	—	General	Specs	Low	No	High	Two Towers PEPA Workb. TIPP Tool
M12	3.3	Petri Net	GSPN	—	General	Specs	Low	No	High	HiQPN GreatSPN DSPNexpress2000
M13	3.4	UML Diagrams	Simulation	—	Real-Time Systems	Design	High	Yes	High	SMG and OPNET
M14	3.4	UML Sequence	Simulation	—	General	Design	High	No	High	SimML, JavaSim
M15	3.5	UML Activity State Diagrams	Semi-Markov Process	—	General	Design	High	No	High	DSPNexpress2000

Fig. 3. Summary of the methodologies.

M11 and **M12** fall into a different group since they apply at software specification time. All the required information to carry out performance analysis has to be interpreted in terms of action/transition execution time. Thus, either all the performance information is available at specification time or the specification has to be properly refined in order to reflect further design choices.

Approach	Reference in Section 3	Req. Analysis	Requirement Specification	Software Architecture	Software Specification	Interface and Component Design	Data Structure and Alg. Design
M1	3.1.1	Performance Requirements	Operational Profile	Execution Env., Scenarios Frequencies		Process View, Service Rates, Resource Contention Delay	Resource Usage, Loop Repetitions, HW Configuration
M2	3.1.1	Performance Requirements	Operational Profile	Execution Env., Scenarios Frequencies		Service Rates	Resource Usage,Loop Repetitions, HW Config., Occurrence Time of Interaction
M3	3.1.1	Performance Requirements	Operational Profile	Execution Env., Scenarios Frequencies, Time of Interaction		Service Rates	Resource Usage, Loop Repetitions, HW Configuration,
M4	3.1.1	Performance Requirements	Operational Profile			Module-Platform Mapping	Platform Configuration, Resource Capacity
M5	3.1.2	Performance Requirements	Operational Profile	Workload			Resource usage, Loop Repetitions, HW Configuration
M6	3.1.2	Performance Requirements	Operational Profile	Workload			HW Configuration
M7	3.1.3	Performance Requirements		Implementation Scenario Alternatives			
M8	3.1.3	Performance Requirements		Critical Performance Scenarios		Env. & Deployment Info.	HW Configuration
M9	3.1.3	Performance Requirements	Number of Calls & Loop Iterations, Sw Comp. to Devices Mapping, Fork Probabilities, Devices Speed-up Factor, System Workload Intensity, Default Values	Number of Calls & Loop Iterations, Sw Comp. to Devices Mapping, Devices Speed-up Factor, Fork Probabilities, System Workload Intensity, Default Values			
M10	3.1.4	Performance Requirements		Classes for Resources		Workload, Running Config., Scheduling Policy for resources	Service Demand
M11	3.2	Performance Requirements			Action Execution Time		
M12	3.3	Performance Requirements			Transition Execution Time		
M13	3.4	Performance Requirements		Temporal Information		Temporal Information	
M14	3.4	Performance Requirements				Performance Information	
M15	3.5	Performance Requirements				Service Time Distributions & Deterministic Delays	

Fig. 4. Performance information required by the methodologies.

The last three methodologies, **M13**, **M14**, and **M15**, propose different approaches based on simulation techniques and stochastic processes, respectively. The first two generate the simulation models based on architectural and design artifacts enriched with temporal information. In order to produce the stochastic process model, the last needs very detailed information at design level, like associating

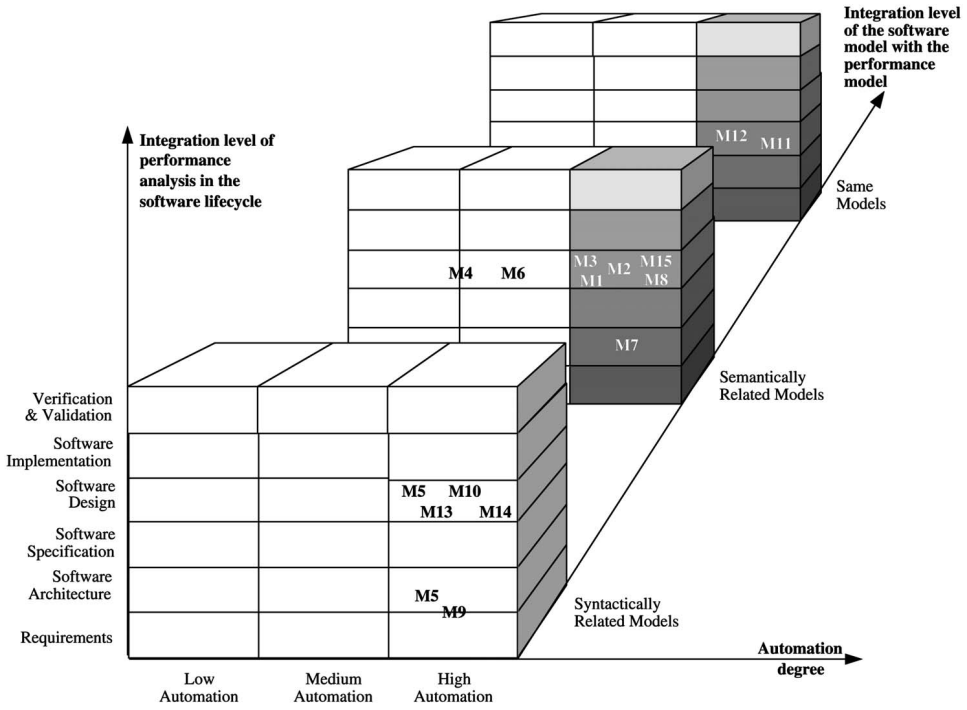

Fig. 5. Classification of considered methodologies.

duration and delays to events. This implies that, at design level, there must be a clear picture of the performance effects of high-level design concepts.

- Few approaches provide feedback information. In general, this requires a direct correspondence between the software specification abstraction level and the performance model evaluation results. Note that the existence of a unique behavioral and performance model does not necessarily imply the existence of this correspondence. Let us consider a process algebra specification, whose syntax provides the description of the software structure in terms of processes and their composition, and the behavioral and performance evaluation is carried out the global state transition model. Like in traditional debugging approaches, it is possible to build a direct correspondence between the syntactic and the execution level, but it can be highly complex and costly. Obviously, methods based on simulation techniques can more easily provide feedbacks because there can be a direct correspondence between the software specification abstraction level and the performance model evaluation results.
- Most approaches exhibit high automation potentiality. This clearly indicates that there is a strong concern about automation as a key factor for the applicability of the approach. In general, automation is considered at any level of the methodology, from the specification of the software artifacts to the derivation of performance model to the evaluation phase. It is worth noting that there is a common trend toward achieving automation through integrating with existing tools both at software specification level, e.g., UML CASE tools, and at the performance analysis level, e.g., QN analyzers.

From this detailed information, we derive the three main indicators introduced in Section 2 to classify the various approaches. The integration level of a software model with the performance model takes into account the architectural constraints and application domain. The integration of performance analysis in the software life cycle accounts for life cycle phase, information level, and feedback. Automation is kept as an autonomous dimension which also comprises tool support.

According to the discussion in Section 2, in Fig. 5, we classify the methodologies. In each internal small cube, label ordering has no meaning. Differently, when a label lies at the intersection of more cubes, like **M4**, this indicates that the corresponding methodology has characteristics of both classes.

Most of the methodologies fall in the groups of syntactically or semantically related models and in the layer referring to software design. Let us recall that the dimension concerning the integration level of software models with the performance models refers to the way the mapping between software models and performance models is carried out. In the group of syntactically related models, we put the methodologies that integrate the design and the performance models based on structural-syntactical mappings. For example, in **M9**, there is an explicit correspondence between syntactic elements at the design level and elements of the performance model, e.g., UCM components and LQN tasks.

The group of semantically related models refers to methodologies whose mapping between design and performance models is mainly based on the analysis of the dynamic behavior of the design artifacts. For example, in **M8** the mapping is based on the analysis of the state machines model of the system design.

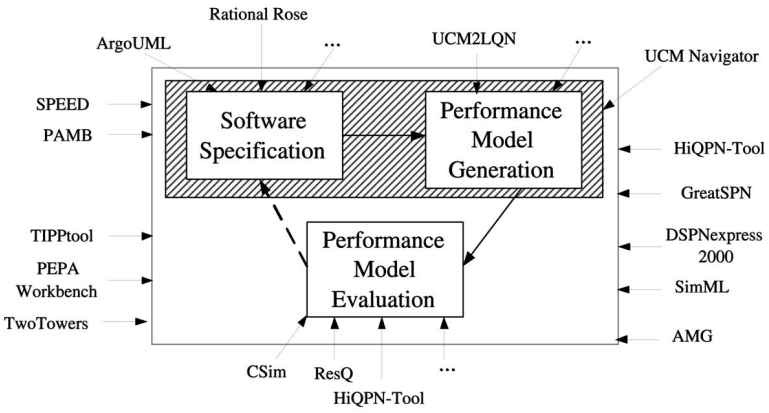

Fig. 6. Tools and performance process.

The last group singles out the methodologies that allow the use of the same model both for design description and performance analysis. This group can actually be seen as a particular case of the semantically related models when the mapping is the identity modulo timing information. However, we have decided to keep it separate because this ideal integration can be obtained only if the software designer has the skill to work with sophisticated specification tools.

The analysis of the integration level of software models with the performance models shows from a different perspective the tendency to use separate formalisms and/or models. As we already pointed out, the rationale for this choice is to offer software designers tools at the same level as their expertise and skills in order to increase the acceptance and the use of the methods. The same reason motivates the high automation degree offered by most of the methods.

The integration level of performance analysis in the software life cycle shows that most of the methods apply to the software design phase in which a good approximation of the information needed to performance analysis is usually provided with a certain level of accuracy. At design level, many crucial design decisions of the software architecture and programming model have already been taken and it is thus possible to extract accurate information for performance analysis, e.g., communication protocols and network infrastructure, process scheduling. At a higher abstraction level of the software system design, there are many more degrees of freedom to be taken into account that must be suitably approximated by the analyst, thus making the performance analysis process more complicated. Of course there is a trade off here and approaches operating at different abstraction levels rather than conflicting can be seen as complementary since they address different needs. Performance prediction analysis at the very early stages of design allows for choice from different design alternatives. Later on, performance analysis serves the purpose of validating such choices with respect to nonfunctional requirements.

Another consideration embracing all the dimensions regards the *complexity* of the methods as far as their ability to provide a performance model to be evaluated is concerned. In this respect, we consider the complexity of the translation algorithms to build the performance model and the information to carry out the analysis. This notion of complexity does not consider the complexity of the analysis itself, although there can be a trade off between complexity of deriving the performance model and complexity of evaluating the model. For example, highly parameterized models can be easily obtained but very difficult to solve. Of course, the efficacy of a methodology and, therefore, its success also depends on the analysis process, thus this issue should be taken into account as well. We do not consider it now since an accurate analysis of the complexity of the analysis process depends on a set of parameters that can require extensive experimentation and use of the methodologies and this kind of analysis is out of the scope of the present paper.

Roughly speaking, with respect to the three dimensions of Fig. 5, the possible methods which occur at the bottom of the second group, on the leftmost side of the automation dimension, exhibit a high complexity. This complexity decreases when moving out along any of the three dimensions. This is due to different reasons. Toward syntactically related models or toward the same models, the mapping between behavioral and performance models simplify or disappears. Moving up along the integration level of performance analysis into the software life cycle dimension, the accuracy of available information increases and simplifies the production of the performance model.

The last issue we discuss concerns the automation of the methodologies and in particular tries to summarize the portions of the software performance process that have been most automated.

Fig. 6 shows the three stages of any software performance analysis process and highlights some automation tools that can be applied in these stages. A tool can support one or more stages as indicated by the arrow entering the stage or the box enclosing the set of stages. The dashed arrow going from performance model evaluation to software specification represents the analysis of potential feedback. It is dashed since, so far, not all the tools pointing to the largest box automatically support this feature.

The picture shows that most of the tools apply to the whole process providing a comprehensive environment. On the other hand, there exist several tools with different characteristics that automate single stages of the process.

This might suggest the creation of an integration framework where the tools automating single stages of the process can be plugged in, in order to provide a best-fit analysis framework.

Among the tools cited in Fig. 6, there exist commercial tools and academic tools. In particular, most of the tools applying to the whole software performance process have been developed by academic institutions. Some of them are just prototype tools (even not available, in some cases), while others do have a commercial version, like GreatSPN, TwoTowers, and DSPNexpress 2000. The tool SPE•ED is the only purely commercial tool. The correspondence between the considered methodologies and their supporting tools is shown in the last column of Fig. 3. We refer to the presentation of the various methodologies for a brief description of the main characteristics of their supporting tools.

5 Conclusions

In this paper, we have reviewed the state of the art in model-based software performance prediction. We have taken a software-designer perspective in order to classify and evaluate the growing number of approaches that have lately appeared. Our choice is driven by the generally acknowledged awareness that the lack of performance requirement validation in current software practice is mostly due to the knowledge gap between software engineers/architects and quality assurance experts rather than due to foundational issues. Moreover, short time to market requirements make this situation even more critical. In this scenario, expressive and intuitive notations, coupled with automated tools for performance validation, would allow quantitative numerical results to be interpreted as design feedback, thus supporting quick and meaningful design decisions.

This is confirmed by the classification we carried out. Referring to Fig. 5, it clearly appears that almost all methodologies try to encompass the whole software life cycle starting from early software artifacts. It is also meaningful that most methodologies are tightly coupled with tool support that allows the (partial) automation of them. There is no methodology which is fully supported by automated tools and, at the same time, there is no methodology that does not provide or foresee some kind of automatic support. Most approaches try to apply performance analysis very early, typically at the software architecture level. So far, most of them still require much detailed information from the implementation/execution scenarios in order to carry out performance analysis. Nevertheless, there is a growing number of attempts that try to relax implementation/execution constraints in order to make the analysis applicable at abstract design levels.

Three indications are highlighted from this survey. The first concerns software design specifications, the second performance models, and the third one is related to the analysis process.

For software design specifications, we believe that the trend will be to use standard practice software artifacts, like UML diagrams. Queuing Networks and their extensions are candidates as performance models. QN provide an abstract/black-box notation, thus allowing easier feedback and model comprehension, especially in a component-based software development process. As far as the analysis process is concerned, besides performance analysis, feedback provision is a key success factor for a widespread use of these methodologies.

Obviously, several problems still remain to be studied and solved. Software notations should allow for easily expressing performance related attributes and requirements. The more abstract the software notation, the more difficult it is to map the performance model. For QN, this is a problem that can only be amended by the existence of algorithms and tools that permit the creation of performance models from standard software artifacts. This is not a problem for GSPN and SPA provided that the designers use these same notations for the behavioral descriptions, which, in software practice, is rarely the case. Therefore, if we assume a standard development process, with standard software artifacts, like UML-based ones, the effort to produce a performance model from the behavioral description is comparable for all three models.

Another problem concerns the complexity of the obtained performance model. This sometimes might prevent efficient model evaluation. The existence of a strong semantic mapping between the software artifacts and the performance model may suggest strategies to reduce the performance model complexity still maintaining a meaningful semantic correspondence. Complexity problems can also be addressed by using simulation techniques besides analytical ones.

Last but not least, the whole analysis process asks for automatic support. How to provide useful feedback is an important issue that deserves more study and experiments.

Summarizing, we believe that, from this survey, it emerges that although no comprehensive methodology is at present available, the field of model-based software performance prediction is mature enough for approaches that can be profitably put into practice.

Acknowledgments

This work was partially funded by MIUR FIRB Project PERF and Project SAHARA.

References

[1] F. Andolfi, F. Aquilani, S. Balsamo, and P. Inverardi, "Deriving Performance Models of Software Architectures from Message Sequence Charts," *ACM Proc. Second Int'l Workshop Software and Performance,* pp. 47-57, 2000.

[2] D. Austry and G. Boudol, "Algébre de Processus et Synchronization," *Theoretical Computer Science,* vol. 30, no. 1, pp. 91-131, 1984.

[3] M. Ajmone, G. Balbo, and G. Conte, *Performance Models of Multiprocessor Performance.* MIT Press, 1986.

[4] F. Aquilani, S. Balsamo, and P. Inverardi, "Performance Analysis at the Software Architecture Design Level," *Performance Evaluation,* vol. 45, no. 4, pp. 205-221, 2001.

[5] L.B. Arief and N.A. Speirs, "A UML Tool for an Automatic Generation of Simulation Programs," *ACM Proc. Second Int'l Workshop Software and Performance,* pp. 71-76, 2000.

[6] P. Buchholz, "A Framework for the Hierarchical Analysis of Discrete Event Dynamic Systems," PhD thesis, Univ. of Dortmund, Germany, 1996.

[7] F. Baccelli, G. Balbo, R.J. Boucherie, J. Campos, and G. Chiola, "Annotated Bibliography on Stochastic Petri Nets," *Performance Evaluation of Parallel and Distributed Systems—Solution Methods,* CWI Tract, 105, pp. 1-24, 1994.

[8] G. Balbo, S. Bruell, and S. Ghanta, "Combining Queueing Networks and Generalized Stochastic Petri Nets for the Solution of Complex Models of System Behaviour," *IEEE Trans. Computers,* vol. 37, pp. 1251-1268, 1988.

[9] S. Balsamo, M. Bernardo, and M. Simeoni, "Combining Stochastic Process Algebras and Queueing Networks for Software Architecture Analysis," *ACM Proc. Int'l Workshop Software and Performance,* pp. 190-202, 2002.

[10] R.J.A. Buhr and R.S. Casselman, *Use CASE Maps for Object-Oriented Systems.* Prentice Hall, 1996.

[11] M. Bernardo, P. Ciancarini, and L. Donatiello, "ÆMPA: A Process Algebraic Description Language for the Performance Analysis of Software Architectures," *ACM Proc. Int'l Workshop Software and Performance,* pp. 1-11, 2000.

[12] L. Bass, P. Clements, and R. Kazman, "Software Architecture in Practice," *SEI Series in Software Eng.,* 1998.

[13] J. Banks, J.S. Carson II, B.L. Nelson, and D.M. Nicol, *Discrete-Event System Simulation.* Prentice-Hall, 1999.

[14] M. Bernardo, W.R. Cleaveland, and W.S. Stewart, "TwoTowers 1.0 User Manual," http://www.sti.uniurb.it/bernardo/twotowers/, 2001.

[15] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni, "Model-Based Performance Prediction in Software Development: A Survey," Technical Report TR 011/2004, Dipartimento di Informatica, Univ. di L'Aquila, 2004.

[16] S. Bernardi, S. Donatelli, and J. Merseguer, "From UML Sequence Diagrams and Statecharts to Analysable Petri Net Models," *ACM Proc. Int'l Workshop Software and Performance,* pp. 35-45, 2002.

[17] M. Bernardo and R. Gorrieri, "A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time," *Theoretical Computer Science,* vol. 202, pp. 1-54, 1998.

[18] J. Bosch and P. Molin, "Software Architecture Design: Evaluation and Transformation," *Proc. 1999 IEEE Eng. of Computer Systems Symp.,* pp. 4-10, Dec. 1999.

[19] H. Beilner, J. Mäter, and C. Wysocki, "The Hierarchical Evaluation Tool HIT," *Proc. Seventh Int'l Conf. Modelling Techniques and Tools for Computer Performance Evaluation,* 1994.

[20] V. Cortellessa, A. D'Ambrogio, and G. Iazeolla, "Automatic Derivation of Software Performance Models from CASE Documents," *Performance Evaluation,* vol. 45, pp. 81-105, 2001.

[21] P.S. Coe, F.W. Howell, R.N. Ibbett, and L.M. Williams, "Technical Note: A Hierarchical Computer Architecture Design and Simulation Environment," *ACM Trans. Modelling and Computer Simulation,* vol. 8, no. 4, pp. 431-446, 1998.

[22] V. Cortellessa, G. Iazeolla, and R. Mirandola, "Early Generation of Performance Models for Object-Oriented Systems," *IEE Proc.-Software,* vol. 147, no. 3 pp. 61-72, 2000.

[23] V. Cortellessa and R. Mirandola, "Deriving a Queueing Network Based Performance Model from UML Diagrams," *ACM Proc. Int'l Workshop Software and Performance,* pp. 58-70, 2000.

[24] "C++Sim,"http://cxxsim.ncl.ac.uk/, 1997.

[25] "CSIM—Performance Simulator,"http://www.atl.lmco.com/proj/csim, 2004.

[26] M. DeMiguel, T. Lambolais, M. Hannouz, S. Betgé-Brezetz, and S. Piekarec, "UML Extensions for the Specification and Evaluation of Latency Constraints in Architectural Models," *ACM Proc. Int'l Workshop Software and Performance,* pp. 83-88, 2000.

[27] P.H. Enslow Jr., "What is a 'Distributed' Data Processing System?" *Computer,* vol. 11, no. 1, pp. 13-21, Jan. 1978.

[28] G. Franks, A. Hubbard, S. Majumdar, D.C. Petriu, J. Rolia, and C.M. Woodside, "A Toolset for Performance Engineering and Software Design of Client-Server Systems," *Performance Evaluation,* vol. 24, nos. 1-2, pp. 117-135, 1995.

[29] R. Franks and C.M. Woodside, "Performance of Multi-Level Client-Server Systems with Parallel Service Operations," *ACM Proc. Int'l Workshop Software and Performance,* pp. 120-130, 1998.

[30] S. Gilmore and J. Hillston, "The PEPA Workbench: A Tool to Support a Process Algebra-Based Approach to Performance Modelling," *Proc. Seventh Int'l Conf. Modelling Techniques and Tools for Performance Evaluation,* pp. 353-368, 1994.

[31] N. Götz, U. Herzog, and M. Rettelback, "TIPP—A Language for Timed Processes and Performance Evaluation," Technical Report 4/92, IMMD7, Univ. of Erlangen-Nürnberg, Germany, 1992.

[32] H. Gomaa and D.A. Menascé, "Design and Performance Modeling of Component Interconnection Patterns for Distributed Software Architectures," *ACM Proc. Int'l Workshop Software and Performance,* pp. 117-126, 2000.

[33] H. Gomaa and D. Menascé, "Performance Engineering of Component-Based Distributed Software Systems," *Performance Eng.,* R. Dumke et al., eds. pp. 40-55, 2001.

[34] V. Grassi and R. Mirandola, "PRIMAmob-UML: A Methodology for Performance Analysis of Mobile Software Architectures," *ACM Proc. Int'l Workshop Software and Performance,* pp. 262-274, 2002.

[35] G. Gu and D.C. Petriu, "XSLT Transformation from UML Models to LQN Performance Models," *ACM Proc. Int'l Workshop Software and Performance,* pp. 227-234, 2002.

[36] C.A.R. Hoare, *Communicating Sequential Processes.* Prentice-Hall, 1985.

[37] J. Hillston, "PEPA—Performance Enhanced Process Algebra," Technical Report CSR-24-93, Dept. of Computer Science, Univ. of Edimburgh, UK, 1993.

[38] F. Hoeben, "Using UML Models for Performance Calculation," *ACM Proc. Int'l Workshop Software and Performance,* pp. 77-82, 2000.

[39] P.G. Harrison and J. Hillston, "Exploiting Quasi-Reversible Structures in Markovian Process Algebra Models," *Computer J.,* vol. 38, no. 7, pp. 510-520, 1995.

[40] H. Hermanns, U. Herzog, and J.P. Katoen, "Process Algebra for Performance Evaluation," *Theoretical Computer Science,* vol. 274, nos. 1-2, pp. 43-87, 2002.

[41] U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle, "Compositional Performance Modelling with the TIPPtool," *Performance Evaluation,* vol. 39, nos. 1-4, pp. 5-35, 2000.

[42] J. Hillston and N. Thomas, "Product Form Solution for a Class of PEPA Models," *Performance Evaluation,* vol. 35, no. 3, pp. 171-192, 1999.

[43] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computations.* Addison Wesley, 1979.

[44] ITU-Telecommunication Standardization Sector, "Message Sequence Charts," ITU-T Recommentation Z. 120(11/99), 1999.

[45] JavaSim, http://javasim.ncl.ac.uk/, 1998.

[46] L. Kleinrock, *Queueing Systems, Volume 1: Theory.* Wiley, 1975.

[47] K. Kant, *Introduction to Computer System Performance Evaluation.* McGraw-Hill, 1992.

[48] P. Kähkipuro, "UML-Based Performance Modeling Framework for Component-Based Distributed Systems," *Proc. Performance Eng. Conf.,* pp. 167-184, 2001.

[49] P. King and R. Pooley, "Derivation of Petri Net Performance Models from UML Specifications of Communication Software," *Proc. 25th UK Performance Eng. Workshop,* 1999.

[50] J.G. Kemeny and J.L. Snell, *Finite Markov Chains.* Springer, 1976.

[51] C. Lindemann, A. Thümmler, A. Klemm, M. Lohmann, and O.P. Waldhorst, "Performance Analysis of Time-Enhanced UML Diagrams Based on Stochastic Processes," *ACM Proc. Int'l Workshop Software and Performance,* pp. 25-34, 2002.

[52] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik, *Quantitative System Performance: Computer System Analysis Using Queueing Network Models.* Prentice-Hall, 1984.

[53] R. Milner, *Communication and Concurrency.* Prentice-Hall, 1989.

[54] D.A. Menascé, "A Framework for Software Performance Engineering of Client/Server Systems," *Proc. 1997 Computer Measurement Group Conf.,* 1997.

[55] D.A. Menascé and H. Gomaa, "On a Language Based Method for Software Performance Engineering of Client/Server Systems," *ACM Proc. Int'l Workshop Software and Performance,* pp. 63-69, 1998.

[56] D.A. Menascé and H. Gomaa, "A Method for Design and Performance Modeling of Client/Server Systems," *IEEE Trans. Software Eng.,* vol. 26, no. 11, pp. 1066-1085, Nov. 2000.

[57] OPNET Manuals, Mil 3, Inc., 1999.

[58] *Developer 5.1 Reference Manual,* ObjecTime Ltd., Ottawa, Ont., 1998.

[59] R. Pooley, "Using UML to Derive Stochastic Process Algebra Models," *Proc. 25th UK Performance Eng. Workshop,* pp. 23-34, 1999.

[60] R. Pooley and P. King, "The Unified Modeling Language and Performance Engineering," *Proc. IEE Software,* pp. 2-10, 1999.

[61] Petri nets tools database, http://www.daimi.au.dk/PetriNets, 2004.

[62] D.C. Petriu and H. Shen, "Applying UML Performance Profile: Graph Grammar-Based Derivation of LQN Models from UML Specifications," *Proc. Seventh Int'l Conf. Modelling Techniques and Tools for Performance Evaluation,* pp. 159-177, 2002.
[63] D.C. Petriu and C.M. Woodside, "Software Performance Models from System Scenarios in Use Case Maps," *Proc. 12th Int'l Conf. Modelling Tools and Techniques for Computer and Comm. System Performance Evaluation,* pp. 141-1158, 2002.
[64] D.C. Petriu and X. Wang, "From UML Descriptions of High-Level Software Architectures to LQN Performance Models," *Proc. Applications of Graph Transformations with Industrial Relevance Workshop (AGTIVE'99),* pp. 47-62, 1999.
[65] W. Reisig, "Petri Nets: An Introduction," *EATCS Monographs on Theoretical Computer Science,* vol. 4, 1985.
[66] J. Rambaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen, *Object-Oriented Modeling and Design.* Prentice-Hall, 1991.
[67] J.A. Rolia and K.C. Sevcik, "The Method of Layers," *IEEE Trans. Software Eng.,* vol. 21, no. 8, pp. 682-688, 1995.
[68] C.U. Smith, *Performance Engineering of Software Systems.* Addison Wesley, 1990.
[69] M. Sereno, "Towards a Product Form Solution for Stochastic Process Algebras," *Computer J.,* vol. 38, no. 7, pp. 622-632, 1995.
[70] C.U. Smith and L.G. Williams, *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software.* Addison Wesley, 2002.
[71] C.U. Smith and L.G. Williams, "Performance Engineering Evaluation of Object-Oriented Systems with SPE●ED℠," *Springer LNCS 1245,* pp. 135-153, 1997.
[72] K.S. Trivedi, *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* John Wiley and Sons, 2001.
[73] C.M. Woodside, C. Hrischuk, B. Selic, and S. Brayarov, "Automated Performance Modeling of Software Generated by a Design Environment," *Performance Evaluation,* vol. 45, pp. 107-123, 2001.
[74] C.M. Woodside, J. Neilson, S. Petriu, and S. Mjumdar, "The Stochastic Rendezvous Network Model for Performance of Synchronous Client-Server-Like Distributed Software," *IEEE Trans. Computer,* vol. 44, pp. 20-34, 1995.
[75] *Proc. ACM Int'l Workshop Software and Performance,* 1998.
[76] *Proc. ACM Int'l Workshop Software and Performance,* 2000.
[77] *Proc. ACM Int'l Workshop Software and Performance,* 2002.
[78] L.G. Williams and C.U. Smith, "Performance Evaluation of Software Architectures," *Proc. ACM Int'l Workshop Software and Performance,* pp. 164-177, 1998.
[79] L.G. Williams and C.U. Smith, "PASA: A Method for the Performance Assesment of Software Architectures," *Proc. ACM Int'l Workshop Software and Performance,* pp. 179-189, 2002.
[80] "Unified Modeling Language (UML)," version 1.5, OMG Documentation, http://www.omg.org/technology/documents/formal/ uml.htm, 2003.
[81] Object Management Group, "UML Profile for Schedulability, Performance, and Time," OMG document ptc/2002-03-02, http://www.omg.org/cgi-bin/doc?ptc/2002-03-02, 2002.
[82] "Extensible Markup Language (XML) 1.0,"second ed., W3C Recommendation 6, Oct. 2000, http://www.w3.org/TR/2000/REC-xml-20001006.

Simonetta Balsamo is a professor of computer science at the University Cá Foscari of Venice (Italy). Her research interests include performance and reliability modeling and analysis of computer and communication systems, parallel and distributed processing, and distributed simulation. She has recently studied quantitative analysis of software architectures and integrated approaches of specification languages with software architecture performance evaluation models and methods. She has published several papers in international journals, conference proceedings, book chapters, and editions of special issues of journal and international conferences. She is member of the ACM.

Antinisca Di Marco received the degree in computer science at the University of L'Aquila in 2001. She is a PhD student in computer science at University of L'Aquila. Her main research interests are: software architecture, software performance analysis and integration of functional and nonfunctional software validation.

Paola Inverardi is a full professor at the University of L'Aquila. Previously, she has worked at IEI-CNR in Pisa and at Olivetti Research Laboratory in Pisa. She is the head of the Department of Computer Science at University of L'Aquila, where she leads the Software Engineering and Architecture Research Group. Her main research area is in the application of formal methods to software development. Her research interests primarily concentrate in the field of software architectures. She has actively worked on the verification and analysis of software architecture properties, both behavioral and quantitative for component-based, distributed, and mobile systems. She has served as general chair, program chair, and program committee member for many international conferences. She is a member of the IEEE.

Marta Simeoni received the PhD degree in computer science at the University La Sapienza of Rome in 2000, and joined the University Cá Foscari of Venice in the same year. She is an assistant professor in the Computer Science Department at the University Cá Foscari of Venice.