Modeling and Analysis of Non-functional Properties in Component-based Systems

Antonia Bertolino [1]

Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", CNR
Pisa, Italy

Raffaela Mirandola [2]

Dipartimento di Informatica, Sistemi e Produzione
University of Roma TorVergata
Roma, Italy

Abstract

This paper discusses methodologies for the specification and analysis of performance related properties of components and assemblies of components, and outlines an original approach, called the CB-SPE, for component-based software performance engineering. The proposed approach relies on, and adapts to a CB framework, the concepts and steps of the SPE technology, and uses for modeling the standard RT-UML profile, reshaped according to the CB principles.

1 Introduction

In recent years the way software systems are designed and built is undergoing great changes at fast pace. Two are the main emerging trends today: the *component-based paradigm* of production, whereby systems are made by assembling pre-existing components; and a progressive *upward shift of developer's focus* in the direction of the abstract levels of system conception and architecture specification.

Our research relies on, and attempts to merge, both trends, in that we are working at a general framework for enabling the early validation of component-based (CB) systems on the basis of the architectural specification. Our anal-

[1] Email:bertolino@iei.pi.cnr.it
[2] Email:mirandola@info.uniroma2.it
[3] This research is carried on within the Pisatel Lab under a cooperation agreement with Ericsson Lab Italy (Rome) and it is partially supported by MIUR-COFIN project SAHARA: Software Architecture for heterogeneous access network infrastructures.

ysis encompasses both the functional and non-functional qualities of a system, and this paper focuses on the approach we are pursuing for the latter, and specifically for performance-related parameters [4]. Before presenting our approach for performance analysis of CB systems, let us briefly discuss the relevant features of the two above mentioned trends.

The intuitive and attractive idea of obtaining complex systems by the rapid assembly of simpler components captivates industrial practitioners with the promise of higher reliability and maintainability at lower costs. However, such advantages can be obtained only via a rigorous design discipline and by accepting standard modelling notations as well as strict documentation and design rules, so that components independently built can effectively be connected and properly interact.

This basic notion is central to the Design-by-Contract discipline [10], originally conceived for Object-Oriented systems , but even better suited for CB development (indeed Objects and Components, though differing concepts, share many aspects). In the same manner that no electrical engineer would produce an electrical circuit without an accompanying specification of the device properties, so a software engineering should provide precise specifications of a component functional and non-functional properties, as made explicit by [17] in his often quoted definition: *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only.* Bachmann et al. [1] go further stating that in a CB system: (i) components and frameworks should have certified properties; and (ii) these certified properties should provide the basis for predicting properties relative to the whole system built out of those components.

We distinguish between properties concerning functional aspects, which guarantee adequate behavior ("correct components"), and non-functional properties, such as reliability and performance, which ensure useful components. The latter form the Quality-of-Service (QoS) level of a component contract. In particular, our focus here is on performance specifications, which are essential in the context of CB production for two basic reasons [14]:

(i) Multiple component implementations may provide the same functional behavior, and the clients will choose those components that best fit their performance requirements.

(ii) If components have performance specifications, then the performance of the system can be derived directly based on its components, while the component implementations need not be re-analyzed in each new context where they are used.

We observe that generality and adaptability of components, which are desirable for reusability, can be detrimental to performance properties and that a CB system generally performs worse than a correspondent optimized

[4] A companion paper [5] discusses a related approach for testing functional properties.

system developed from scratch. Therefore, the ability to predict as early as possible the QoS characteristics of the assembled system becomes crucial to ascertain whether the CB product will satisfy its requirements, or rather a different, more specialized process should be adopted.

We now come to the second current tendency in system engineering. While the need emerges of abiding by the contracts, at the same time the level of abstraction at which those contracts should be established is rising. In fact, to ensure interoperability across components built with different technologies and implemented for different environments, researchers and producers have understood that the room for negotiation and combination of properties is at the level of architectural specifications, at which specific implementation details can be abstracted away. High-level and standardized models must hence be adopted, in such a way that the compatibility and the interactions among components can be early verified.

The widespread adoption of the UML evidences this trend, and its flexibility to specialized yet standard-compatible extensions, where necessary, provides a valuable tool for pursuing this direction. Cheesman and Daniels [6] describe how UML can be specialized for modelling within a CB paradigm embracing the basic principles of the above discussed Design-by-Contract approach. Very recently, the OMG Model Driven Architecture (MDA) approach to development pursues a complete separation between the base platform-independent model of an application, and the descriptions of one or more platform-specific models, describing how the base is implemented on each of the supported platforms [11]. The base model in MDA is specified in UML.

In this direction, our intended contribution here is twofold: we provide a brief overview of current work on performance analysis of CB systems, and we propose a general framework in which we aim at putting together the most recent advances in the fields of: (i) CB software engineering, (ii) Software Performance Engineering (SPE) and (iii) UML modelling of CB system architectures. Our final goal is enabling the analysis of performance properties on the assembled system from the models of the components and of the overall architecture.

In the next two sections we provide some background information about the technologies we use, namely SPE and RT-UML; in Sect. 4 we overview related work, providing a fresh look at this just starting field; in Sect. 5, we lay out an outline of the approach we are building towards CB performance engineering; finally, we draw conclusions in Sect. 6.

2 Software Performance Engineering

Software Performance Engineering (SPE), firstly presented in [15], is a systematic, quantitative approach to constructing software systems that meet performance objectives. It is based on the careful and methodical assessment of performance issues throughout the lifecycle, from requirements and specifi-

cation to implementation and maintenance.

The SPE basic concept is the separation of the Software Model (SM) from its environment or Machinery Model (MM). This distinction, on the one hand, allows for defining software and machinery models separately and solving their combination, on the other improves the portability of the models (e.g., the performance of a specific software system can be evaluated on different platforms, and the performance of a specific platform can be validated under different software systems).

The SPE process consists of the following steps [16,13]:

1 Assess Performance Risk: the level of risk and its impact on system performance determines the amount of effort to put into SPE activity.

2 Identify Critical Use Cases, i.e., those use cases that are mostly important to responsiveness or scalability for the user(s) of the system

3 Select Key Performance Scenarios, i.e., those that are executed frequently or that are perceived as critical to the performance

4 Establish Performance Objectives, i.e., for each key performance scenario specify quantitative criteria for evaluating its performance characteristics and the conditions (workload mix and intensity) under which the performance objective should be achieved.

5 Construct Performance Models (as above explained).

6 Determine Software Resource Requirements, i.e., the amount of processing and software resources required for each scenario step.

7 Add Computer Resource Requirements, i.e., the load imposed on the devices used by scenario steps. Computer resource requirements depend on the environment in which the software executes.

8 Evaluate Performance Models: using the model and the selected analysis method, compute the performance predictions. If feasible: choose the most promising design approach; otherwise, if infeasible, change product requirements.

These activities proceed in parallel with the evaluation of the models themselves.

3 Real-Time UML

While the UML is generally recognized as a very useful tool for modelling the functional characteristics of a system, it lacks quantifiable notions of time and resources usage, which has impeded in the past its broader adoption in the real-time and embedded domains.

The UML Profile for Schedulability, Performance and Time (RT-UML) has been proposed as a response to these exigences by a working consortium of OMG member companies, and has been recently adopted as an OMG stan-

dard [18]. RT-UML is not an extension to the UML metamodel, but a set of domain profiles for UML allowing for the construction of models that can be used to make (early in the life cycle) quantitative predictions regarding the characteristics of timeliness, schedulability, and performance. The RT-UML profile was not conceived for a specific analysis method, but is intended to provide a single unifying framework encompassing the existing analysis methods, still leaving enough flexibility for different specializations. Basically, the underlying idea is to import as annotations in the UML models the characteristics relative to the target domain viewpoint (performance, real-time, schedulability, concurrency), in such a way that various (existing and future) analysis techniques can usefully exploit the provided features.

The profile is partitioned into a number of *sub-profiles*, i.e., "profile packages dedicated to specific aspects and analysis techniques". Here we focus on the performance analysis (PA) sub-profile.

A performance context specifies one or more scenarios, i.e., ordered sequences of steps, describing various dynamic situations involving the usage of a specified set of both processing and passive resources under specified workloads (i.e., the load intensity and the required or estimated response times for the scenario). A scenario might involve multiple concurrent threads due to forking within a scenario. In particular, PA scenarios can be modeled following either a Collaboration-based approach or an Activity-based approach.

4 Overview of related work

In this section we present a short survey of the existing work on performance evaluation of CB systems and on performance modeling using UML and RT-UML based software models. We treat these areas in two separate subsections below, also because to the best of our knowledge, ours is the first attempt to merge them into a unifying framework, as we describe in the next section.

4.1 Performance of CB software systems

There are not many works on CB performance engineering, due both to the novelty of the topic and to its inherent difficulty. When the emphasis is on the quantitative evaluation of performance, the approach mostly applied is measurement [7,22]. In [22] a discussion is brought of how component-based system properties may influence the selection of methods and tools used to obtain and analyze performance measures. Then a method is proposed for the measurement of performance distinguishing between application-specific metrics (e.g., execution time of various functions) and platform-specific metrics (e.g., resource utilization). The automation of the process of gathering and analysing data for these performance metrics is also discussed. The major drawbacks of this approach is that it is suitable only for already implemented systems and moreover the obtained results do not show general applicability.

A different approach that partially overcomes these difficulties is presented in [7] where starting from a specific COTS middleware infrastructure, a first step empirically collecting performance measures is followed by a second step in which the obtained results are elaborated to extend their validity to a more general setting. The proposed approach also includes: a reasoning framework for understanding architectural trade-offs and the relationship with technology features; and the derivation of a set of mathematical models describing the generic behavior of applications using that specific COTS technology. An inherent limitation of this approach is that it leads to sound results only for a specific platform.

A different approach targeted to include predictability in performance behavior of CB systems is presented in [14]. The basic idea is that the "behavior of a CB system must be compositional in order to be scalable" and this requires (as in our approach) to include also performance specifications in addition to descriptions of component functional behavior. The paper outlines how classical techniques and notations for performance analysis are either unsuitable or unnatural to capture performance behaviors of generic software components, and explains that "performance specification problems are so basic that there are unresolved research issues to be tackled even for the simplest reusable components". A first attempt towards a compositional approach to performance analysis is then presented mainly based on the use of formal techniques. However, as the authors claim, an engineering approach to predictability on performance is a necessary ingredient to ensure predictable components.

Finally, a different, mainly qualitative, approach to the performance predictability/analysis of CB systems is undertaken in [20,2], where the affinity between software architecture (SA) and software component (SC) technology is outlined and exploited. This affinity is related to different aspects: (i) the central role of components and connectors as abstraction entities, (ii) the correlation of architectural style and component model and frameworks, (iii) the complementary agendas followed by the SA and SC technologies: enabling reasoning about quality attributed, and simplifying component integration. Therefore, the basic idea underlying these works is to develop a reference model that put in relations the key abstractions of SA and CB technology and then to adapt and apply some existing SA analysis methods, such as ATAM, QADP,etc.

4.2 Performance from UML and RT-UML-based models

In the last years several papers have presented methodologies for the generation of performance evaluation models starting from UML diagrams, possibly augmented with suitable performance annotations ([8], papers in [21]). These papers consider different target models, spanning simulation models, Petri nets and queueing networks. Recently, the growing interest in SAs has brought to extending performance model generation to also encompass the SA concept,

with particular emphasis given to the impact on performance of the software organization into components and patterns of interaction (papers in [21]). In all these papers the target performance model is a Queueing Network.

As already stated, the adoption as a standard of the PA profile (see Section 2.2) is quite recent and therefore there are very few papers dealing with RT-UML based software systems. Selic in [18,13] provides a general framework (and relative guidelines) for the automatic performance model generation starting from RT-UML diagrams. A first attempt to use the recently adopted standard *UML Performance Profile* is presented in [12]. This paper proposes a graph grammar-based method for the automatic transformation of a UML model annotated with performance information into a Layered Queueing network (LQN) performance model. The LQN structure is generated from the high level SA showing the architectural patterns used in the system, and from Deployment Diagrams indicating the allocation of software components to hardware devices. The LQN model parameters are derived from information relative to key performance scenarios.

A completely different application of the PA profile has been proposed in [3,4], where performance engineering is applied to aid the project manager in making schedule predictions and in optimizing personnel utilization during software development. In this context the project teams are assimilated to the processing elements of a performance model, and their activities to the tasks to be accomplished within established time intervals. Then by use of the proposed approach different workflow assumptions can be explored and their consequent outcomes automatically derived. A front-end interface based on a subset of Real-Time UML is provided to allow people not expert of performance modeling notations to apply the approach.

5 CB-SPE: the proposed approach

As already stated, our long term goal is to define a general framework for quality validation of CB systems, and in this paper we focus on performance related properties. Examples of performance attributes we tackle are system and components execution times, response times, resource utilization and so on. The basic idea is to follow the SPE approach also for CB development, i.e.: to introduce the definition and the validation of the performance attributes of interest since the early specification and design stages so to achieve in the end both components and CB applications that "guarantee" specific performance requirements. The proposed approach is called the CB-SPE.

In the application of SPE to CB analysis, our intuition suggests that the separation between the SM and the MM can be quite naturally mapped within the MDA or similar approaches, whereby the SM parallels the base platform independent model, and the MM refers to the information relative to the platform specific models.

The CB-SPE approach involves both usage layers:

1 *The component layer*: that guarantees to have components with certified performance properties

2 *The application layer*: that guarantees to have CB applications with the required performance.

For each, we briefly discuss the main involved issues and present an example when the software modeling notation is based on RT-UML.

Component layer: In this layer the goal is to obtain components with certified performance properties that are explicitly declared in the component interfaces. This implies that the component developer must introduce and validate performance requirements from the early stage of component specification and design to the final stages of component implementation and deployment by following the SPE approach. Although component performance properties strongly depend on the execution environment of the component itself, our goal is to have component performance properties that are, in some sense, platform independent (thus they are valid for every environment), but also able to take into account the platform characteristics, as in the MDA development process. So we define, for each development stage, a component model that includes an abstract but quantified specification of the environment, so obtaining a model for the specific performance environment rather than for a specific platform. Let us suppose, for example, that the performance index we are interested in is the CPU demand of service for each offered service S, then the annotation could be of the form $CPU - demand(S(env - par))$, where $env - par$ denotes the environment parameters.

Then, according to the SPE process, once defined the key performance objectives, we can validate the model by instantiating the values obtained with the platform independent approach against those of a concrete platform [5] to see if a given environment can support the model. In other words, at the component level, we would obtain a model that explicitly takes into account the kind of expected (compatible) environment, yet is still independent of any specific technology (like in [13]). The obtained performance results for the various services offered by the component are finally exported at the component interfaces.

RT-UML example: As said in Section 3, we use the RT-UML PA profile, that allows for accurate specification of key performance aspects directly on UML models. For the component concepts, we refer to [6] where a component is defined as a set of interrelated component forms, each reflecting some aspects of a component during the development lifecycle: component specification, interface, implementation, installed component and component

[5] The level of detail at which it is possible to model the executing platform mainly depends on the available information, however some abstraction could be necessary to obtain a tractable model.

object. Each form can be modeled by a set of UML diagrams as described in [6]. Our idea is that in CB-SPE the component developer can annotate these diagrams following the PA profile. Examples of annotations are the response time (required or measured) of a specific behavior/service, or the host execution demand that in this case will be parametric, the probability to execute a given service and so on [18]. Then, for each offered service/behavior, starting from the RT-UML component specification it is possible, following one of the approaches proposed in the literature [8,3,4,12,13], to (automatically) derive performance models that are SMs with parametric resource requirements. These results should then be exported in the component interface and annotated following the RT-UML syntax.

Application layer: Goal of this step is to obtain CB applications with the expected performance properties, by applying the SPE approach. So the system assembler, to realize its application, chooses among the available components those that better fulfill the performance requirements.

In fact, at this step, the system assembler knows the characteristics of the platform to which the component are deployed and thus he/she can instantiate the component performance properties to the given environments. The various component performance properties can now be combined in a model by following the "logic" of the application. If, from the model analysis, we conclude that in such a way the performance requirement are fulfilled we can proceed with the acquisition of the components and their assembly, otherwise we have to continue searching by repeating these steps.

A very attractive feature of the CB-SPE approach is that it naturally can support a compositional analysis of performance properties in that we can progressively integrate and calculate the performance parameters of interest as components are added to the system.

RT-UML example: As for the component layer we use as application modeling notation the RT-UML and require that the system assembler describes the application logic through one or more sequence diagrams (SDs) (activity diagrams could also be used). In the CB framework the SD objects represent the components involved, and the SD messages represent the requests of execution of a component service or correspond to information/data exchanged between the components. Moreover the system assembler should construct a Deployment Diagram (DD) modeling the resources available and their characteristics. In this case the nodes of the DD can be associated to: classical resources (device, processor, database) and communication means.

We note that the system assembler does not have to repeat this step from scratch each time he/she needs to make estimations about an application. The same diagrams can be re-used for similar applications, by only updating the associated parameters, as described here below.

The SD and DD diagrams have to be annotated with the proper perfor-

mance values and parameters. The system assembler should express, by using a comment-based annotation, the attributes associated with events and actions of the diagrams [6].

For example, considering the SD, classical examples of PA attributes are: the population, which represents the number of jobs in the scenario,the response time, which represents the application completion time and is one of the expected results. Considering the DD nodes, examples of PA attributes concern the resource scheduling policy that models the strategy by which the resource handles the different jobs, the resource utilization, the resource throughput that represents the amount of work provided per unit of time by a resource belonging to a certain node.

As for the component layer, once the application has been modeled using the RT-UML diagram it is possible, by applying the methods proposed in the literature [8,3,4,12,13], to automatically derive a complete performance model.

6 Conclusions and future work

This paper presents a starting point towards an engineering approach to encompass performance validation in CB systems on the basis of the architectural specification. We have outlined an original approach, called the CB-SPE, that relies on, and adapts to a CB framework, the concepts and steps of the well-known SPE technology and uses for modeling the standard RT-UML profile, reshaped according to the CB principles.

Future work includes the detailed definition of the methodology and its application to case studies coming from the industrial world.

References

[1] Bachman F., et al., "Technical Concepts of Component-Based Software Engineering", Tech. Rep. CMU/SEI-2000-TR-008.

[2] Bass L., Klein M., Bachman F., "Quality attributes design primitives" Technical note CMU/SEI- 2000-TN-017.

[3] Bertolino A., Marchetti E., Mirandola, R., *Real-Time UML-based Performance Engineering to Aid Manager's Decisions in Multi-project Planning*, in [21].

[4] Bertolino A., Lombardi G., Marchetti E., Mirandola R., *Software Performance Measures to Assist Decision Makers within the Rational Unified Process*, Proc. of 12th IWSM 2002, Magdeburg, Germany, October 2002.

[5] Bertolino A., Marchetti E., Polini A., *Integration of "Components" to Test Software Components*, in these Proceedings.

[6] In particular, the PA attributes of the closed workload will be associated with the first action of the SD; those of the steps will be linked to each of the subsequent message activations; those of the resources will be related to each of the nodes of the DD.

[6] Cheesman J., Daniels J., "UML Components, A simple process for specifying component-based software", Addison Wesley, 2001

[7] Chen S., Gorton I., Liu A., Liu Y., *Performance prediction of COTS Component-based Enterprise Applications*, Proc. ICSE Workshop 5th CBSE 2002.

[8] Cortellessa V., Mirandola R., *PRIMA-UML: a Performance Validation Incremental Methodology on Early UML Diagrams*, Science of Computer Programming **44** (2002), 101-129, July 2002, Elsevier Science.

[9] Lavenberg S.S., "Computer Performance Modeling Handbook", (New York, 1983), Academic Press.

[10] Meyer B., "Applying 'Design by Contract'", Computer **25** (10), Oct. 1992, 40-52.

[11] "Model Driven Architecture, A Technical Perspective" doc. n. ab/2001-01-01, n. ormsc/2001-07-01

[12] Petriu D.C., Shen H., *Applying the UML Performance Profile: Graph Grammar-based derivation of LQN models from UML specification*, Proc. of Performance TOOLS 2002, London, England, April 14-17 2002, LNCS 2324, Springer Verlag.

[13] Selic B. "Performance-Oriented UML capabilities", Tutorial talk in [21]

[14] Sitaraman M., Kulczycki G., Krone J., Ogden W., Reddy A.L.N., *Performance specification of software components*, Proc. of SSR '01, p. 310, ACM/SIGSOFT, May 2001.

[15] Smith, C.U. "Performance Engineering of Software Systems", Addison-Wesley, 1990.

[16] Smith, C.U. and L. Williams. "Performance Solutions: A practical guide to creating responsive, scalable software", Addison-Wesley, 2001.

[17] Szyperski C., "Component Software: Beyond Object-Oriented Programming", Addison Wesley, 1998.

[18] "UML Profile for Schedulability, Performance, and Time Specification, On line at `http://cgi.omg.org/docs/ptc/02-03-02.pdf`

[19] "UML Documentation version 1.4 Web Site", On-line at `http://www.rational.com/uml/resources/documentation/`

[20] Wallnau K., et al., *On the relationship of software architecture to software component technology*, Proc WCOP6, 2001

[21] Proceedings of the Third Int. Workshop on Software and Performance WOSP2002, ACM, 2002.

[22] Yacoub S., *Performance Analysis of Component-Based Application*, Proc. of the 2nd Software Product Line Conference, p.299-315.