# Parametric Performance Contracts for Software Components and their Compositionality

Ralf H. Reussner, Viktoria Firus, and Steffen Becker

Software Engineering Group, OFFIS, Escherweg 2, D-26121 Oldenburg
University of Oldenburg
[reussner|firus|becker]@informatik.uni-oldenburg.de

**Abstract.** The performance of a software component heavily depends on the environment of the component. As a software component only justifies its investment when deployed in several environments, one can not specify the performance of a component as a constant (e.g., as a single value or distribution of values in its interface). Hence, classical component contracts allowing to state the component's performance as a post-condition, if the environment realises a specific performance stated in the precondition, do not help. This fixed pair of pre- and postcondition do not model that a component can have very different performance figures depending on its context. Instead of that, parametric contracts are needed for specifying the environmental dependency of the component's provided performance. In this paper we discuss the specification of such dependencies for the performance metric response time. We model the statistical distribution of response time in dependency of the distribution of response times of environmental services.

## 1 Introduction

Performance is an issue for many computer systems. While formerly performance was a major concern primarily in the domain of real-time control systems, nowadays performance became an issue in the domain of enterprise information systems. Although information systems rarely depend on adhering to strict real-time constraints, the importance of response-times, reaction-time, throughput and scalability is obvious, in particular, in B2C eCommerce systems, where unsatisfactory performance directly translates into financial losses.

*Component-based* software architectures offer one benefit for performance predictions: the compositional structure of the software can be reflected in compositional performance prediction models. These models aim at predicting the system's performance according to the architecture used and the performance of the components deployed [1–3]. But even the existence of *compositional performance models* does not solve the entire problem of performance prediction. This is because, one also needs *component performance models* in order to model the performance of a single component in dependency of its environment [4].

The latter point is most critical: as a component depends on its environment by calls to external services, also its performance is influenced by the environment's performance. Consequently, component performance data measured in one specific envi-

ronment can not be used for predictions of component performance in different environments.

The contribution of this paper is modelling the dependency of component performance on the component's context by compositional component performance models. We argue, that performance models (like any model for predicting QoS of component-based software) have to be (a) compositional, (b) parametric, and (c) precise. We therefore propose a component performance model based on parametric contracts. In this model the response time (or other linear additive metrics, such as reaction time) is specified by random variables.

The paper is organised as follows. In Section 2 we introduce component contracts, and the concept of parametric contracts. We briefly introduce the Quality of Service Modelling Language (QML) and stochastic basics we draw upon. Section 3 begins with the general discussion of dependency models and defines parametric performance contracts. We also state assumptions and limitations of the model. There, we also make explicit which values our model needs and discuss where or how to gather them. Section 4 presents related work to our work. Section 5 concludes and presents open issues. In particular, we discuss refinements and extensions of our performance model and the relaxation of assumptions.

## 2 Fundamentals

### 2.1 Parametric Contracts

We model the contractual use of a software component at design- or reconfiguration-time as follows [5]: the requires interface represents the pre-condition of the component, as it describes the conditions the component expects its environment to fulfil in order to operate. The provides interface represents the post-condition of the component, as it describes the services the environment can expect the component to offer, if the pre-condition is met by the environment (This corresponds to Meyer design-by-contract principle [6], but is lifted from methods to components).

However, if quality attributes are included in interface descriptions, this single pair of pre- and post-conditions is insufficient as it does not model the dependency of a component's quality attributes (such as reliability or performance) on its context [4] [1]. Therefore, we need to model the dependency between the context's quality attributes and the component's quality attributes depending on those. On the functional level we found service effect automata [7] useful. A *service effect automaton* is a finite state machine, describing for each service implemented by a component, the set of possible sequences of calls to services of the context. Therefore, a service effect automaton is a control-flow abstraction. Control-statements (if, while, etc.) are neglected, unless they concern calls to the component's context. As an example, consider the following code (see figure 1, left hand side) and its associated service effect automaton (see figure 1, right hand side):

[1] Although less obvious, even the mere functionality to be provided by a component depends on the context [7].

```
void DoOrderBilling(ListOfOrders orders,
                    CCServer myCCServer)
{
    myCCServer.Connect(
               resources.GetCCServerURL());

    foreach (Order o in orders)
    {
        if (!o.HasValidCC())
        {
            BillCashOnDeliver(order);
        }
        else
        {
            myCCServer.Transfer(order);
        }
    }
    myCCServer.Disconnect();
}
```

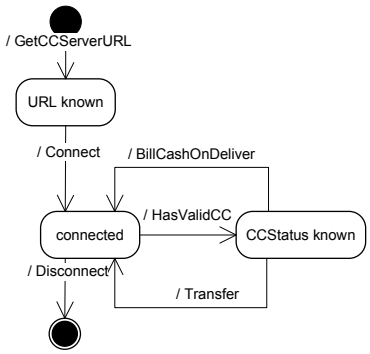


**Fig. 1.** Payment example

It can be seen, that transitions correspond to external calls, while any internal computation is abstracted away within nodes. Due to that, service effect automata are more abstract as the component implementation.

Parametric contracts can be used to specify components on the following levels:

**Type level:** As service effect automata are an abstraction of the actual implementation, many different implementations can fulfil a single service effect specification. Therefore, components specified by service effect automata can be used to describe components with their dependencies (as grey-boxes [8]) on the type-level.

**Implementation instance level:** As a service effect specification can be derived automatically out of a component implementation, service effect automata can be used to describe components on a component implementation instance level.

**Run-time instance level:** We separate this level from the previous, as an instantiating a component for execution (i.e., by loading it into the main memory) differs from the above instantiation of a component (type) by choosing or realising a specific implementation of this type. The specification capabilities of the current version of service effect automata for component run-time instances are rather limited. For example, the execution of the control-flow can be modelled if execution is strictly sequential. But as soon as threads are forked or joined, the use of Petri-Nets (or a different calculus) instead of finite state machines seems reasonable.

However, for modelling the dependency between contextual quality attributes and the quality attributes of the component, service effect automata have to be extended. In [9, 10] this was done by an extension to Markov-Models for predicting the reliability of component services.

In section 3 we describe an extension for linear performance metrics, such as response or reaction time.

### 2.2 QML

The Quality of Service Modelling Language (QML) [11] is used to specify QoS attributes for interfaces. It is based on the concepts of contract types and contracts. Con-

tract types are used to specify the metrics used to determine a specific QoS concept. Contracts are used afterwards to specify a certain level of the metrics of a contract type offered or required by a certain interface method. The binding between contracts and interface methods is thereby done via QML profiles.

With respect to our work it is important how the metrics are specified. In QML you can specify mean value, variance and/or percentiles of a metric's distribution. It is not designed for using mathematical distributions like exponential or Gaussian distribution. Therefore if one needs such mathematical distributions one has to estimate a distribution type and its parameters from the given percentiles with mathematical methods.

### 2.3 Random Variables

A random variable $X$ is a measurable function assigning a real number to an outcome of a random experiment. A distribution function of the continuous random variable $X$ is defined as

$$F_X(x) = P(X \leqslant x) = \int_{-\infty}^{x} f_X(x)dx, \qquad -\infty < x < \infty,$$

The function $f_X(x)$ is called density function of $X$.

Two random variables $X$ and $Y$ are independent, if for all $x$ and $y$ the events $\{X \leqslant x\}$ and $\{Y \leqslant y\}$ are independent, i.e.

$$P(X \leqslant x, Y \leqslant y) = P(X \leqslant x)P(Y \leqslant y)$$

.

The density function of the sum of two independent random variables is given by

$$f_{X+Y}(x) = \int_{0}^{x} f_X(z)f_Y(x-z)dz =: f_X \circledast f_Y(x)$$

the so-called convolution of $f_X$ and $f_Y$.

## 3 Modelling Component Performance with Parametric Contracts

For extra-functional properties, the application of parametric contracts is crucial. For example, one can not specify the timing behaviour of a software component as a fixed number. Much more, the timing properties of a component as offered in its provides-interface is always a function of the environment's timing behaviour, as received at its requires-interfaces.

### 3.1 Model

Here we describe the extension of service effect automata to parametric contracts for performance. Each transition (i.e., call to an external method) and each node (internal computation) of a service effect automaton is annotated with a random variable. This

random variable models the time consumption of the call, respective the internal computations. Note, that we assume those random variables to be statistically independent.

Of course, it would be simpler for the analysis to use constants instead of random variables. However, there are several reasons to use random variables. First, the time consumption of internal computations depends on the component's state, i.e., variable values and parameters, which are not modelled by service effect automata. Second, the time consumption of external calls is also not fixed as basically the same as for internal calls holds. Additionally, information systems are commonly not executed on real-time platforms so that there are no timing constraints known from the underlying system services.

### 3.2 Computation of Provides Interfaces

Service effect automata without cyclic dependencies are finite state machines consisting of three basic concepts: sequence, alternative and loop (see figures 2-4).

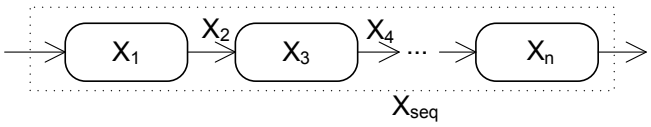


**Fig. 2.** Sequence

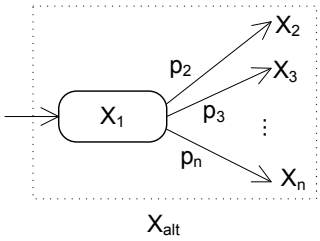


**Fig. 3.** Alternative

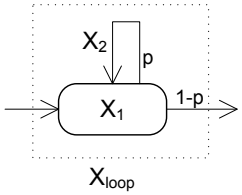


**Fig. 4.** Loop

The timing behaviour of an offered service of a component is characterised by a random variable. As described above this random variable models the time consumption of a call to the offered service. The distribution of the random variable is calculated from the service effect automata of the offered service. By using the nodes and edges of the service effect automata and the density functions associated to them we determine the distribution of the time consumption. Therefore we need to identify the afore mentioned basic concepts in the service effect automata. Afterwards these basic concepts can be calculated as follows.

For a sequential execution of services the time consumption of the whole **sequence** is the sum of the time consumptions for each external call plus the time consumption for internal calculations. Therefore the random variable associated to a call sequence is represented as a sum of the random variables assigned to the individual nodes and edges (see figure 2). The density function results from the convolution of the single density

functions:

$$f_{X_{seq}}(x) = \overset{n}{\underset{i=1}{\circledast}} f_{X_i}.$$

A random variable associated to an **alternative** is represented as a sum of the alternative paths weighted with the call probabilities (see figure 3). The associated density function has the following form:

$$f_{X_{alt}}(x) = f_{X_1} \circledast (\sum_{i=2}^{n} p_i f_{X_i})(x).$$

A **loop** is either run again with probability $p$ or left with probability $1-p$. Therefore one can represent a loop as a choice of an infinitely amount of alternative paths. The associated random variable has the following form:

$$X_{loop} = \begin{cases} X_1 & \text{with probability} \quad 1-p \\ X_1 + (X_2 + X_1) & \text{with probability} \quad (1-p)p \\ \vdots & \\ X_1 + \underbrace{(X_2 + X_1)... + (X_2 + X_1)}_{n} & \text{with probability} \quad (1-p)p^n \\ \vdots & \end{cases}$$

Starting from a certain $n$ the probabilities $(1-p)p^n$ become small enough to be neglected. Thus the density function can be represented as a finite sum:

$$f_{X_{loop}}(x) = \sum_{i=0}^{n} (1-p)p^i f_{X_1} \circledast \underbrace{(f_{X_2} \circledast f_{X_1}) \circledast ... \circledast (f_{X_2} \circledast f_{X_1})}_{i}.$$

For the payment example (see figure 1) let $p_1$ be the probability for calling `HasValidCC` and $p_2$ be the probability for calling `BillCashOnDeliver`. Then the density function of the random variable associated to `DoOrderBilling` has the following form:

$$\begin{aligned} f_{DoOrderBilling}(x) = f_{GetCCServerURL} \circledast f_{URLknown} \circledast f_{Connect} \circledast f_{connected} \circledast \\ (\sum_{i=0}^{n} (1-p_1)p_1^i \overset{i}{\underset{}{\circledast}} f_{loop} \overset{i}{\underset{}{\circledast}} f_{connected} \circledast f_{Disconnect})(x), \end{aligned}$$

$$f_{loop} = f_{HasValidCC} \circledast f_{CCStatusknown} \circledast (p_2 f_{BillCashOnDeliver} + (1-p_2) f_{Transfer})$$

### 3.3 Compositionality

By compositionality we refer to the fact, that the result of the evaluation of a parametric contract (i.e., a specific pre- or post-condition) can be used as an input for other parametric contracts. This enables us to chain evaluate parametric contracts, i.e., to connect components and to consider the assembly as a component again.

Whether our model is compositional in the above sense, depends on the distribution used and whether one is willing use approximations. For example, if exponential distributions are used for modelling the service time of external calls, one can compute the distribution of the service effect specification as given above. However, the resulting distribution need not have to be an exponential distribution itself. For the sake of compositionality, one can approximate the result by an exponential distribution again. Hence, there is a trade-off between precision and compositionality on our model. Further investigations will have to evaluate this.

### 3.4 Which values do we need and where do we get them from?

For our model we need distributions of random variables modelling method call performance. Further on we need the service effect automata of all services involved in the calculation. Annotated to the service effect automata we need probabilities of the branches in alternatives and loop constructions.

The needed information can be gathered by different approaches. First, code analysis can yield the service effect automata and probability values. Additionally probability values can be gathered by measuring a usage profile or simply by guessing.

Further on specialised methods for performance engineering, i.e., SPE [12], use specific models of the system to predict the system's performance. These models can also be used as information source for our approach.

## 4 Related Work

SPE is one of the first approaches which provides a technique for evaluating the performance of software systems [12]. That approach can be enhanced if specialised for component based software engineering [13]. The basic idea is to use already known performance information of the pre-produced components. The components‘ environment and usage profile are included in the calculation explicitly in the proposed approach.

In our approach we don't take into account neither usage profile nor input data yet. The dependency on the input data is addressed in [14]. This article proposes an approach that allows to specify performance contracts parametrised by input values of objects.

In [15] probability concepts are applied to performance evaluation of computer systems. Among other things distributions of random variables are utilised to calculate timing behaviour of whole applications.

A common terminology for the prediction of Quality of Service of systems assembled from components is proposed in [16]. A concrete methodology for predicting .NET assemblies is presented in [1].

As mentioned before QML can be used to specify QoS contracts for component interfaces. There is also a specialised variant of QML called CQML [17] designed for specifying component oriented QoS. It introduces the specification of exact mathematical distributions as well as the idea of compositional reasoning over several CQML contracts. Nevertheless the compositional reasoning is immature in CQML.

## 5 Conclusion and Future Work

In this paper we discuss an extension of parametric contracts with performance aspect. Therefore we demonstrate parametric performance contracts and their application. We also show that these contracts are compositional. An empirical evaluation is necessary and can be done either by comparing predicted and monitored performance values or at least by checking the correctness of the design decisions were the performance prediction lead to. Questions to be answered in such empirical validations are:

**Model properties, such as compositionality or precision:**
As model compositions can get more and more complex every time one composes them again, it remains an open question which kind of complexity can be handled analytically by our approach. In case of infeasible complexity, the mathematical analysis can be simplified considerably by using mean value and variance of the computation times instead of random variables.

**Mathematical assumptions:** It has to be investigated how the assumption of the independence of the random variables can be relaxed. This is important when considering several components hosted on the same machine.

**Model assumptions:** Assumptions like modelling internal computations by random variables must be justified (or abandoned in favour for simpler models using constants).

In general we are interested in finding out whether a simulation or mixed approach (instead of a pure analytical model) can be taken in order to estimate the distribution functions associated to the random variables. [2]

#### Acknowledgements

We thank Jens Happe for valuable discussions, his implementation of the Palladio component model and, in particular for pointing out to us the different instance levels of a component.

## References

1. Dumitrascu, N., Murphy, S., Murphy, L.: A methodology for predicting the performance of component-based applications. In Weck, W., Bosch, J., Szyperski, C., eds.: Proceedings of the Eighth International Workshop on Component-Oriented Programming (WCOP'03). (2003)
2. Balsamo, S., Simeoni, M.: Deriving performance models from software architecture specifications. In: Proceedings of the 15. European Simulation Multiconference 2001 (ESM 2001), Prague. (2001)
3. Petriu, D.C., Wang, X.: From UML description of high-level software architecture to LQN performance models. In Nagl, M., Schürr, A., Münch, M., eds.: Proc. Applications of Graph Transformations with Industrial Relevance, International Workshop, AGTIVE'99 Kerkrade, The Netherlands, September, 1999. Volume 1779., Springer (2000)

[2] Further details on the Palladio project are available at:
`http://se.informatik.uni-oldenburg.de/research/Palladio`

4. Reussner, R.H.: Contracts and quality attributes of software components. In Weck, W., Bosch, J., Szyperski, C., eds.: Proceedings of the Eighth International Workshop on Component-Oriented Programming (WCOP'03). (2003)
5. Reussner, R.H., Schmidt, H.W.: Using Parameterised Contracts to Predict Properties of Component Based Software Architectures. In Crnkovic, I., Larsson, S., Stafford, J., eds.: Workshop On Component-Based Software Engineering (in association with 9th IEEE Conference and Workshops on Engineering of Computer-Based Systems), Lund, Sweden, 2002. (2002)
6. Meyer, B.: Applying "design by contract". IEEE Computer **25** (1992) 40–51
7. Reussner, R.H.: Automatic Component Protocol Adaptation with the CoCoNut Tool Suite. Future Generation Computer Systems **19** (2003) 627–639
8. Büchi, M., Weck, W.: The greybox approach: When blackbox specifications hide too much. Technical Report 297, Turku Center for Computer Science (1999)
9. Reussner, R.H., Poernomo, I.H., Schmidt, H.W.: Reasoning on software architectures with contractually specified components. In Cechich, A., Piattini, M., Vallecillo, A., eds.: Component-Based Software Quality: Methods and Techniques. Number 2693 in LNCS. Springer-Verlag, Berlin, Germany (2003) 287–325
10. Reussner, R.H., Schmidt, H.W., Poernomo, I.: Reliability prediction for component-based software architectures. Journal of Systems and Software – Special Issue of Software Architecture - Engineering Quality Attributes **66** (2003) 241–252
11. Frølund, S., Koistinen, J.: Quality-of-service specification in distributed object systems. Technical Report HPL-98-159, Hewlett Packard, Software Technology Laboratory (1998)
12. Smith, C.U., Williams, L.G.: Performance Solutions: a practical guide to creating responsive, scalable software. Addison-Wesley (2002)
13. Bertolino, A., Mirandola, R.: Towards component based software performance engineering. In Crnkovic, I., Schmidt, H., Stafford, J., Wallnau, K., eds.: Proc. 6th Workshop on Component-Based Software Engineering: Automated Reasoning and Prediction, ACM/IEEE 25th International Conference on Software Engineering ICSE 2003. (2003) 1 – 6
14. Krone, J., Ogden, W.F., Sitaraman, M.: Modular verification of performance constraints. Technical Report RSRG-03-04, Clemson University (2003)
15. Trivedi, K.S.: Probability and Statistics with Reliability, Queuing and Computer Science Applications. Prentice Hall, Englewood Cliffs, NJ, USA (1982)
16. Hissam, S.A., Moreno, G.A., Stafford, J.A., Wallnau, K.C.: Packaging predictable assembly. In: Proceedings of the IFIP/ACM Working Conference on Component Deployment, Springer-Verlag (2002) 108–124
17. Aagedal, J.Ø.: Quality of Service Support in Development of Distributed Systems. PhD thesis, University of Oslo (2001)