Parametric Performance Contracts for QML-specified Software Components

Viktoria Firus [1] Steffen Becker [2] Jens Happe [3]

Software Engineering Group
University of Oldenburg
26121 Oldenburg, Germany

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 dependencies of external calls for the performance metric response time. We present an approach using parametric contracts to compute the statistical distribution of response time as a discrete distribution in dependency of the distribution of response times of environmental services. We use the Quality of Service Modeling Language (QML) as a syntax for specifying distributions.

Key words: performance prediction, parametric performance contracts, service time distribution

1 Introduction

Performance is an issue for many computer systems. For this reason one is interested in methods to evaluate the performance of a software system during early stages of development, i.e., during architectural design. The main goal of architectural performance analysis is to evaluate design alternatives and to guide the decision between these alternatives.

[1] Email: viktoria.firus@informatik.uni-oldenburg.de
[2] Email: steffen.becker@informatik.uni-oldenburg.de
[3] Email: jens.happe@informatik.uni-oldenburg.de

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

Component-based software architectures offer at least two benefits for performance predictions [1]: (a) The use of pre-fabricated components considerably limits the design and implementations decisions to be made for implementing an architecture. By that limitation of the degrees-of-freedom the performance of the implementation of the architecture becomes more predictable. (b) 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 [2,3,4]. 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 [5]. The latter point is most critical: the component's performance is influenced by calls to external services, the hardware the component is deployed on and the usage profile of the component. Consequently, component performance data measured in one specific environment can not be used for predictions of component performance in different environments.

In [6] a concept of parametric performance contracts was presented, which models the performance of the offered services of the component depending on the performance of external services. The time consumption of the external services and of internal computations of the component is characterised by random variables with continuous distribution functions. This approach suffered from the lack of compositionality and simplicity of expressions of distributed functions.

By compositionality we refer to the fact, that the result of the evaluation of a parametric contract can be used as an input for other parametric contracts. This enables us to chain the evaluation of parametric contracts, i.e., to connect components and to consider the assembly as a component again. Thus, if a specific class of distribution functions is used for modeling the time behavior of external calls, then the result of the parametric contract should belong to the same class. But this is not always the case when using continuous distribution functions.

Additionaly the resulted distribution functions for parametric contracts had a complicated presentation and therefore a limited expressiveness. Approximations with appropriate statistical distributions can be used to cope with the complexity of presentations and the non-compositionality of the resulted functions. But, this would lead to less precision of the performance predictions.

In this paper we discuss the use of discrete distributions to describe the timing behavior of external services. This enables the specification of any time behavior of the component's individual services. We are not dependent on a specific class of statistic distributions. The results of the computations are again discrete functions, i.e. the model is compositional.

The contribution of this paper is modeling 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 paramet-

ric contracts. In this model the response time (or other linear additive metrics, such as reaction time) is specified by random variables. In particular, we discuss the use of discrete performance distributions, because they are simple to specify by the Quality of Service Modelling Language (QML) [7] and their use complies with the above requirements (a) – (c) to performance models.

The paper is organised as follows. In Section 2 we introduce component contracts, and the concept of parametric contracts. We introduce the Quality of Service Modelling Language (QML) and its discrete probability distributions as well as other stochastic basics we draw upon. Section 3 begins with the general discussion of dependency models and defines parametric performance contracts. Section 4 presents related work to our work. Section 5 concludes and presents open issues.

2 Fundamentals

2.1 Parametric Contracts

We model the contractual use of a software component at design- or reconfiguration-time as follows [8]: 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 [9], 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 [5]. 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 [10] 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 code snippet on the left hand side of the figure 1 and its associated service effect automaton on right hand side.

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.

For getting the requires protocol of a component, one has to build the transitive closure to internal calls, as internal calls can themselves call external services. But this is of no concern here. Note that the service effect automata of a component do not have to be specified manually, but can be (a) either generated out of design documents, such as message sequence charts, or (b) derived automatically out of the code.

```
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();
}
```

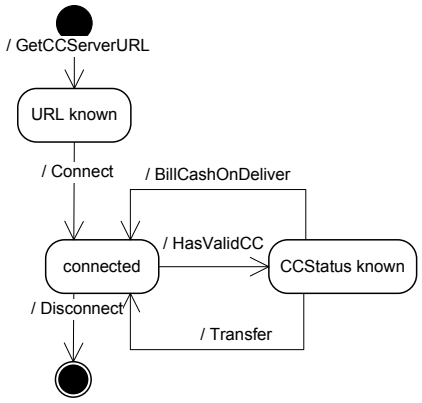

Fig. 1. Payment example

However, for modelling the dependency between contextual quality attributes and the quality attributes of the component, service effect automata have to be extended. In [11,12] 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) [7] is used to specify QoS attributes for interfaces. It is based on the concepts of contract types and contracts. Contract 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. The contract for a quality dimension can be specified by a single value (e.g., $\texttt{no\_transaction\_per\_sec} > 10$). Alternatively, one could specify a distribution of values. (This is useful, as one cannot state the performance of a service with hard upper bounds for response time, and the like, if not using a real-time environment.) As an example, consider the contract shown in figure 2.

This means 100% of the transfer executions must be less than 40 msec. and, as additional constraints, 80% of executions must be less than 20 msec. and 50 % even less than 10 msec. The figure 3 shows the associated discrete distribution function for the values of the performance contract from figure 2.

QML can be used to specify QoS contracts for component interfaces. There is also a specialised variant of QML called CQML [13] 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.

```
from latest require Performance contract {
    transfer {
          percentile 30 < 5 msec;
          percentile 50 < 10 msec;
          percentile 80 < 20 msec;
          percentile 100 < 40 msec;
    };
};
```

Fig. 2. Performance Contract with distribution for values

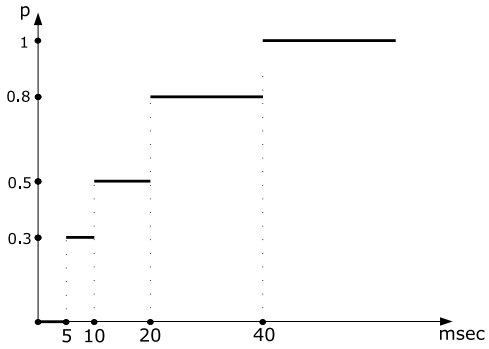

Fig. 3. Distribution Function as defined by QML Performance Contract

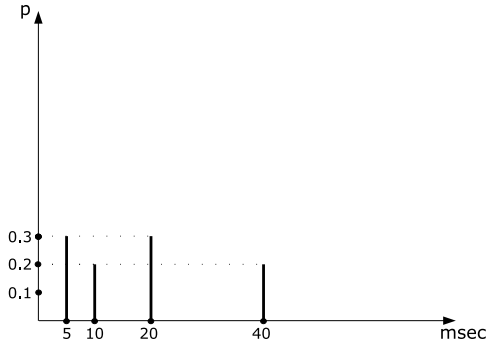

Fig. 4. Probability Mass Function as defined by QML Performance Contract

2.3 Random Variables

We use random variables for extension of the service effect automata with time consumption of internal and external services. A random variable X is a measurable function assigning a real number to an outcome of a random experiment. The probability mass function of a discrete random variable with a finite number m of outcomes is defined as

$$p : x_i \longrightarrow P(X = x_i) \qquad i = 0, \ldots, m-1$$

We also write p_i for $p(x_i)$. Assume the x_i being increasingly ordered. Then the cumulative distribution function of the discrete random variable is defined as following:

$$F(x_k) = P(X \leqslant x_k) = \sum_{i=1}^{k} p_i \qquad k = 0, \ldots, m-1$$

Note, that with probability mass function all the outcomes of the service resp. response times in the interval $]x_{i-1}, x_i], i = 0, \ldots, m-1$ are concentrated on the point x_i. As we intend to use Fourier transform for our calculations, the steps for the probability mass function have to be equidistant. So we have to adapt this function adequately. As no additional information about the distribution is available, we assume the service times in the single intervals being uniformly distributed. Then the stepwidth α can be chosen for example as $\alpha = gcd\{|]x_{i-1}, x_i]|\}_{i=1,2,\ldots,m-1}$, the greatest common divisor of the interval's lengths. We also call α the sampling rate

of the probability mass function. After that each interval $]x_{i-1}, x_i]$ is divided in $\frac{x_i - x_{i-1}}{\alpha}$ subintervals. The probabilities of the points x_i are also uniformly mapped onto new calculated sampling points. As result all the sampling points from the interval $]x_{i-1}, x_i]$ including x_i have the probability $p_{i_{new}} = \frac{p_i \alpha}{x_i - x_{i-1}}$. The figures 5 and 6 show the conversed functions for our QML example.

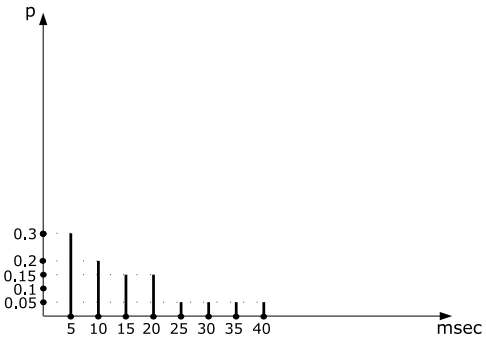

Fig. 5. Equid. Probability Mass Function

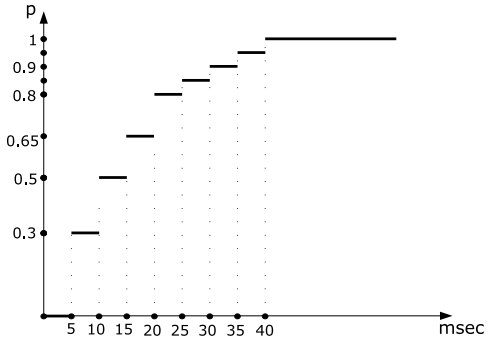

Fig. 6. Equidistant Distribution Function

In the following we write $x_\alpha[n]$ for a probability mass function

$$x_\alpha[n] : n \longmapsto p_n = P(X = \alpha n)$$

with sampling rate α.

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

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

The probability mass function of the sum of two independent random variables X and Y is given by

$$\begin{aligned}(x+y)_\alpha[n] &= P(X + Y = n\alpha) \\ &= \sum_{k=1}^{n} P(X = k\alpha)P(Y = (n-k)\alpha) \\ &= \sum_{k=1}^{n} x_\alpha[k] y_\alpha[n-k] =: x_\alpha[n] \circledast y_\alpha[n]\end{aligned}$$

the so-called convolution of $x_\alpha[n]$ and $y_\alpha[n]$.

In order to calculate the convolution of the probability mass functions we make use of the discrete Fourier transform [14,15]. The discrete Fourier transform of a probability mass function does always exist, since there is an $N > 0$ such that $x_\alpha[n] = 0$ for all $n < 0$ and all $n \geqslant N$ [14].

Next we discuss the calculation of the limit of the expression

$$(1-p) \sum_{k=0}^{\infty} p^k \overset{k}{\underset{l=1}{\circledast}} x_\alpha[n] \tag{1}$$

as we need it later in this paper. A more detailed description can be found in [16]. There, the computation of the limit is given with respect to the approximation of continuous probability density functions. First, consider the Fourier transform of the multiple self convolution:

$$\begin{aligned}\mathcal{F}\left\{\overset{k}{\underset{l=1}{\circledast}} x_\alpha[n]\right\} &= \prod_{l=1}^{k} \mathcal{F}\{x_\alpha[n]\} \\ &= \mathcal{F}\{x_\alpha[n]\}^k\end{aligned}$$

Let $\delta[n]$ be unity impulse [14] which is the neutral element of convolution, that is $\delta[n] \circledast x_\alpha[n] = x_\alpha[n] = x_\alpha[n] \circledast \delta[n]$ for an arbitrary function $x_\alpha[n]$. The unity impulse is defined as

$$\delta[n] = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$

and its discrete Fourier transform is the constant function $\mathcal{F}\{\delta[n]\} = 1$. Therefore, we have for $k = 0$ self convolutions that:

$$\overset{0}{\underset{l=1}{\circledast}} x_\alpha[n] = \mathcal{F}^{-1}\left\{\mathcal{F}\{x_\alpha[n]\}^0\right\} = \mathcal{F}^{-1}\{1\} = \delta[n].$$

Next, we determine the limit of the discrete Fourier transform of the sum:

$$\begin{aligned}\mathcal{F}\left\{\sum_{k=0}^{\infty} p^k \overset{k}{\underset{l=1}{\circledast}} x_\alpha[n]\right\} &= \sum_{k=0}^{\infty} p^k \mathcal{F}\left\{\overset{k}{\underset{l=1}{\circledast}} x_\alpha[n]\right\} \\ &= \sum_{k=0}^{\infty} p^k \mathcal{F}\{x_\alpha[n]\}^k \\ &= \frac{1}{1 - p\mathcal{F}\{x_\alpha[n]\}}\end{aligned}$$

The insertion of the result in equation 1 yields:

$$(1-p)\sum_{k=0}^{\infty} p^k \overset{k}{\underset{l=1}{\circledast}} x_\alpha[n] = (1-p)\mathcal{F}^{-1}\left\{\frac{1}{1 - p\mathcal{F}\{x_\alpha[n]\}}\right\} \tag{2}$$

Since we make no assumptions regarding $x_\alpha[n]$, there are no further simplifications possible. The discrete Fourier transform and its inverse can be efficiently computed using the fast Fourier transform algorithms [15].

Next, we analyse convergence criterion of the function above. During the computation of the limit of the sum in equation 2, we used the convergence of the geometric series:

$$\sum_{k=0}^{\infty} p^k \mathcal{F}\{x_\alpha[n]\}^k = \frac{1}{1 - p\mathcal{F}\{x_\alpha[n]\}}$$

which is only valid for:

$$|p\mathcal{F}\{x_\alpha[n]\}| < 1.$$

In the following we investigate under which conditions the above equation is valid. We can assume that p is greater than zero. If p, the probability of executing the loop, is zero, the loop is never executed and the computation can be omitted. So, only the absolute value of the Fourier transform has to be computed and we have that:

$$p|\mathcal{F}\{x_\alpha[n]\}| < 1.$$

The values of the discrete Fourier transform $\mathcal{F}\{x_\alpha[n]\}$ are:

$$f_j = \sum_{n=0}^{N-1} x_\alpha[n] e^{-\frac{2\pi j}{N} ni} \qquad j = 0, \ldots, N-1$$

where i is the imaginary unit. Inserting this formula into the equation above yields:

$$p|\sum_{n=0}^{N-1} x_\alpha[n] e^{-\frac{2\pi j}{N} ni}| < 1 \qquad j = 0, \ldots, N-1.$$

$x_\alpha[n]$ is a probability mass function with $x_\alpha[n] \geqslant 0$ for all $n \geqslant 0$. So, only the absolute value of the exponential function has to be computed:

$$p\sum_{n=0}^{N-1} x_\alpha[n]|e^{-\frac{2\pi j}{N} ni}| < 1 \qquad j = 0, \ldots, N-1. \tag{3}$$

All complex variables z can be expressed in polar form:

$$z = re^{\omega i},$$

with r as the magnitude of z and ω as the angle of z [14]. If r is one, we have that $z = e^{\omega i}$ lies on the unit circle of the complex plain, and therefore:

$$|e^{\omega i}| = 1$$

for any angle ω. For equation 3 we have that $\omega = -\frac{2\pi j}{N} n$ and $|e^{-\frac{2\pi j}{N} ni}| = 1$ for all $n, j = 0, \ldots, N-1$. This yields :

$$p\sum_{n=0}^{N-1} x_\alpha[n] < 1$$

for every value $j = 0, \ldots, N-1$.

$x_\alpha[n]$ is a probability mass function, whose sum from zero to infinity must be one. Here, $x_\alpha[n]$ is zero for all n greater or equal than N. So, we have that:

$$\sum_{n=0}^{N-1} x_\alpha[n] = 1.$$

Inserting this into the equation above we get:

$$p < 1.$$

Thus the limit for the equation 1 exists for $p < 1$.

3 Modelling Component Performance

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.

(i) The time consumption of internal computations is not fixed. Much more, it depends on the internal state of the components (i.e., the values of its variables and parameters) which is not given by the service effect specification. Even if internal variables and the like were modelled, their actual values are only known at run-time. Hence, for an architectural analysis at design- or reconfiguration-time, this approach would not be suitable anyway.

(ii) The time consumption of external calls is also not fixed. In fact, the time consumption of external services depends on a number of influence factors, such as parameter values, the operating system (as interrupts and scheduling can change service execution times), the network, etc. Again, we do not model these factors, because (a) they are only known at run-time, and (b) they are hard to estimate in advance.

(iii) Information systems are not executed on real-time platforms. Therefore, it is impossible to prove tight bounds of execution times.

An alternative to random variables would be using mean and variance of these computation times. Although this would simplify the mathematical analysis significantly, it basically binds us to the Gaussian distribution of service times. However, service times are usually not Gaussian distributed, as their distribution is not symmetric. This is because, there is always an minimal execution time (bounded by hardware and software architecture) but there exist higher execution times, caused by processing delays.

3.2 Computation of Provides Interfaces

In the following we describe the computation of the probability mass function of a service described by the service effect automaton on the basis of the time behaviour

of external calls and internal services. Service effect automata without cyclic dependencies are finite state machines consisting of three basic concepts: sequence, alternative, and loop (see figures 7-9).

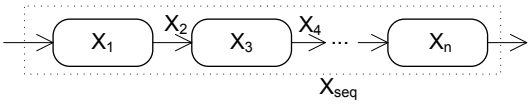

Fig. 7. Sequence

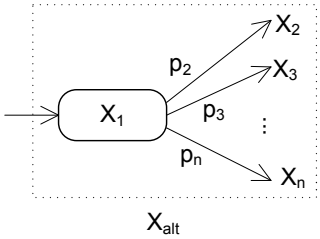

Fig. 8. Alternative

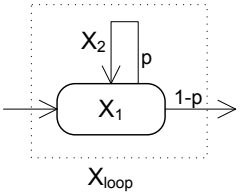

Fig. 9. Loop

The timing behaviour of an offered service of a component is characterised by a random variable. 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 probability mass 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. Note, that all probability mass functions should have the same sampling rate α.

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 7). The probability mass function results from the convolution of the single probability mass functions:

$$x_{seq}[n] = \overset{n}{\underset{i=1}{\circledast}} x_i[n].$$

A random variable associated to an **alternative** is represented as a sum of the alternative paths weighted with the call probabilities (see figure 8). The associated probability mass function has the following form:

$$x_{alt}[n] = x_1[n] \circledast \sum_{i=2}^{n} p_i x_i[n].$$

A **loop** is either run again with probability p or left with probability $1 - p$. There-

fore one can represent a loop as a choice of an infinitely amount of alternative paths. The associated random variable has the following form:

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

The probability mass function for X_{loop} is given by the infinite series:

$$x_{loop}[n] = (1-p) \sum_{k=0}^{\infty} p^k \overset{k}{\underset{l=1}{\circledast}} (x_1[n] \circledast x_2[n]) \tag{4}$$

The limit of the execution time of a loop exists, if and only if the probability of reentering the loop is smaller than one (see equation 2). This corresponds to the intuitive expectation. If the probability is one, the loop would never be left and therefore, the execution time of the loop must be unlimited.

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

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

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

Here, we derived the formula manually from the automaton in figure 1 by identifying the operations sequence, alternative, and loop and joining them according to the automaton. For an automatic analysis the recognition of the required operations and their order is needed. However, this cannot be achieved directly using finite state machines, since cycles cause major problems when deriving the formula from the automaton. Cycles can be interconnect, include other cycles, and have multiple entry points. Therefore, in [16] regular expressions are used to determine the formula of an automaton. Regular expression have the advantage of being hierarchically structured. Hence, the content of a loop and its entry point are clearly specified. Furthermore, regular expressions require exactly the three operations sequence (concatenation), alternative (set union), and loop (Kleene star) as defined above. So, the concept developed here can be transferred to regular expressions relatively easy.

3.3 Computational Complexity

Despite the advantages of discrete functions, the computational complexity might become an issue for functions with a large number of sampling points. The number of sampling points increases with the size of its domain and the accuracy of the sampling. Since the computations used above (convolution, Fourier transform, addition and multiplication of discrete function) have to be applied on each of those sampling points the computational complexity and therefore the execution time of the algorithm depends directly on this number.

Let N be the number of sampling points of the two discrete functions $x_\alpha[n]$ and $y_\alpha[n]$. The computational complexity of the fast Fourier algorithm is $O(N\,log(N))$ for the computation of the discrete Fourier transform and its inverse [15]. The Fourier transform of $x_\alpha[n]$ is a complex discrete function with N values. The complexity of a multiplication/division or an addition/substraction of two functions or of a function and a scalar is $O(N)$, since the operation has to be applied on every sampling point to compute the result. The convolution of two functions becomes a multiplication of their Fourier transforms. Hence, the convolution is not required for the computation of the service time consumption. Given the complexity of all required operations we can determine the total complexity of the algorithm.

For the application of the algorithm, the discrete Fourier transform is computed for the probability mass function of each state and external service of the service effect automaton. All further computations are performed on the Fourier transforms. Finally, the inverse Fourier transform is applied on the result.

Let N_E be the number of external services and N_S the number of states of the automaton. Then, (N_S+N_E+1) is the total number of required Fourier transforms. Therefore, the complexity of the computation for all states, external services, and the result is $O((N_S + N_E + 1)N\,log(N))$. Next, we discuss the complexity of the operations sequence, alternative and loop.

The operation **sequence** for M_S elements requires a M_S-1 convolutions of discrete functions. The convolution becomes a multiplication of the discrete Fourier transform of these functions. Therefore, the complexity of this operation is determined by $O((M_S-1)N)$.

Because of the linearity of the Fourier transform, the computation of the operation **alternative** is the same for the time and frequency space (except the convolution which becomes a multiplication). It requires a scalar multiplication of each alternative with its probability, the addition of the results and a convolution with the function of the origin state. Let M_A be the number of alternatives, then we have M_A multiplications by its probability, M_A-1 additions, and one convolution. This yields a total complexity of $O((M_A + (M_A-1) + 1)N) = O(2M_AN)$.

The Fourier transforms that are required for the computation of the time consumption of a **loop** are omitted here, since all computations are performed in the frequency space. Hence, the computation of the execution time of a loop requires only one subtraction, three multiplications/divisions, and one convolution. This yields a total complexity of $O(5N)$.

Note that all three operations are in the linear complexity class. Then, the total complexity of the algorithm is as follows. Let S, A, and L be the number of sequence, alternative, and loop operations of service effect automata. The total complexity of the algorithm is:

$$O((N_S + N_E + 1)N\,log(N) + SN + AN + LN).$$

The values N_S, N_E, S, A, and L depend on the structure of service effect automata. All these numbers can be assumed to be much smaller than the number of sampling points of the function:

$$N_S, N_E, S, A, L \ll N.$$

Therefore, the computational complexity of the algorithm is in the linearithmic complexity class $O(N\,log(N))$.

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 [17], 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 [17]. That approach can be enhanced if specialised for component based software engineering [18]. 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 [19]. This article proposes an approach that allows to specify performance contracts parametrised by input values of objects.

In [20] 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 [21]. A concrete methodology for predicting .NET assemblies is presented in [2].

5 Conclusion and Future Work

In this paper we discuss an extension of parametric contracts under performance aspects. Particularly, we use discrete distribution functions for modeling the time consumptions of external service calls. Thus, our model is parametric and compositional. To specify the discrete distribution functions of external service calls on the interface level one can use QML. The precision of the computations depends on the chosen stepwidth of the QML specification.

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 our research we aim to incorporate the execution environment and usage profile into the performance prediction of component based software architectures. [4]

References

[1] Becker, S., Firus, V., Giesecke, S., Hasselbring, W., Overhage, S., Reussner, R.: Towards a Generic Framework for Evaluating Component-Based Software Architectures. In Turowski, K., ed.: Architekturen, Komponenten, Anwendungen - Proceedings zur 1. Verbundtagung Architekturen, Komponenten, Anwendungen (AKA 2004), Universität Augsburg. Volume 57 of GI-Edition of Lecture Notes in Informatics., Bonner Köllen Verlag (2004) 163–180

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

[2] 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)

[3] Balsamo, S., Simeoni, M.: Deriving performance models from software architecture specification. In: Proceedings of the European Simulation Multiconference, Analytical and Stochastic Modelling Techniques. (2001)

[4] 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)

[5] 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)

[6] Reussner, R.H., Firus, V., Becker, S.: Parametric performance contracts for software components and their compositionality. In Weck, W., Bosch, J., Szyperski, C., eds.: Proceedings of the 9th International Workshop on Component-Oriented Programming (WCOP 04). (2004)

[7] Frølund, S., Koistinen, J.: Quality-of-service specification in distributed object systems. Technical Report HPL-98-159, Hewlett Packard, Software Technology Laboratory (1998)

[8] 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)

[9] Meyer, B.: Applying "Design by Contract". IEEE Computer **25** (1992) 40–51

[10] Reussner, R.H.: Automatic Component Protocol Adaptation with the CoCoNut Tool Suite. Future Generation Computer Systems **19** (2003) 627–639

[11] 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

[12] Reussner, R.H., Schmidt, H.W., Poernomo, I.H.: Using Parameterised Contracts for Component Reliability Prediction. In Ehrig, H., Krämer, B., Ertas, A., eds.: Integrated Design & and Process Technology. Volume IDPT 1., Society for Process & Design Sciences (2002)

[13] Aagedal, J.Ø.: Quality of Service Support in Development of Distributed Systems. PhD thesis, University of Oslo (2001)

[14] Oppenheim, A.V., Willsky, A.S., Nawab, H.: Signals & Systems, Second Edition. Prentice Hall (1997)

[15] DeFatta, D.J., Lucas, J.G., Hodgkiss, W.S.: Digital Signal Processing: A System Design Approach. John Wiley & Sons, Inc. (1988)

[16] Happe, J.: Reliability prediction of component-based software architectures. Master thesis, Deparment of Computing Science, University of Oldenburg (2004)

[17] Smith, C.U., Williams, L.G.: Performance Solutions: a practical guide to creating responsive, scalable software. Addison-Wesley (2002)

[18] Bertolino, A., Mirandola, R.: CB-SPE Tool: Putting Component-Based Performance Engineering into Practice. In Crnkovic, I., Stafford, J.A., Schmidt, H.W., Wallnau, K.C., eds.: Proc. 7th International Symposium on Component-Based Software Engineering (CBSE 2004), Edinburgh, UK. Volume 3054 of Lecture Notes in Computer Science., Springer (2004) 233–248

[19] Krone, J., Ogden, W.F., Sitaraman, M.: Modular verification of performance constraints. Technical Report RSRG-03-04, Clemson University (2003)

[20] Trivedi, K.S.: Probability and Statistics with Reliability, Queuing and Computer Science Applications. Prentice Hall, Englewood Cliffs, NJ, USA (1982)

[21] Hissam, S.A., Moreno, G.A., Stafford, J.A., Wallnau, K.C.: Packaging predictable assembly. In Bishop, J.M., ed.: Component Deployment, IFIP/ACM Working Conference, CD 2002, Berlin, Germany, June 20-21, 2002, Proceedings. Volume 2370 of Lecture Notes in Computer Science., Springer (2002) 108–124