A Tool Supporting the Analytical Evaluation of Service Level Agreements

Falko Bause, Peter Buchholz, Johannes May
Department of Computer Science
TU Dortmund, 44221 Dortmund
Germany
{falko.bause, peter.buchholz, johannes.may}@tu-dortmund.de

ABSTRACT

Quantitative aspects of modern IT systems are often specified by service level agreements (SLAs) which relate the maximal load of a system with guaranteed bounds for response times and delays. These quantities are specified for single services which are combined in a service oriented architecture (SOA) to composed services offered to potential users or other service providers. To derive SLAs for composed services and to plan the required capacity to guarantee SLAs, appropriate methods and tools have to be used that compute results based on information given in SLAs. In this paper it is argued that most available approaches are not sufficient to analyze systems based on SLA information. A new method and a tool are presented that support the efficient calculation of bounds for delays in composed systems based on bounds for the load and the delay of the individual components which are specified in the SLAs of the components. Furthermore, the presented tool can be used to generate bounds for the required processing capacity which a provider has to provide in order to guarantee the quality of service defined in the SLAs.

The presented approach is in some sense a counterpart to mean value analysis for queueing networks but rather than mean values, worst case bounds for different quantities like response times or departure processes are computed. Analysis is based on min/+ algebra but the mathematical approach is hidden from the user by a graphical interface allowing a simple graphical specification and result representation for networks of composed services.

Keywords

service level agreements, analytical evaluation, performance modeling tools

1. INTRODUCTION

Quantitative aspects of current software systems are specified as part of the service level agreements (SLAs). In a service oriented world, one can distinguish between the user of a service and the provider of the service. The user requests a minimal service level which is guaranteed by the provider. An important part of the service level is the response time. Usually systems guarantee a maximal response time for a large percentage of the submitted load. The provider has to supply the requested processing capacity to process the load within the available time. However, to guarantee maximal response times, the load has to be restricted in the SLA. This implies that an SLA contains an upper bound for the load which the user can submit into the system and an upper bound for the response time to process the submitted load. Apart from these bounds an SLA defines various other quantitative and functional properties of the service, which we will not consider in our analysis. Thus, if we speak of an SLA, we explicitly consider the bounds on arrivals and response times or delays[1].

In many cases, a service is composed of sub-services. This can be the case if the provider uses third party services or if the user composes his or her service from sub-services offered by different providers. In such a case the SLA of the composed service has to be derived from the sub-services. For a provider there is additionally the need to compute the required processing capacity from the knowledge of the SLA to avoid an over- or under-loading of the available equipment.

The definition and analysis of appropriate SLAs is a capacity planning problem. Since the environment in which the problem has to be solved is highly dynamic and often extremely complex, detailed analysis approaches like simulation are not adequate, because the effort would be too high, required data is not or not completely available and some details of the system are unknown. Therefore it is important to develop analysis techniques that can work with the available information which means that the information given in the SLAs is the basis for quantitative analysis and capacity planning.

Abstract models to analyze service oriented architectures (SOAs) are usually based on results available from product form queueing networks. Often some variant of mean value analysis is applied [15–17]. In some cases hierarchical models are supported [21] but analysis is still based on product form techniques. Overview papers describing different approaches to analyze quality-of-service in service oriented architectures or cloud environments are [1, 11].

There are two disadvantages of queueing network based

ICPE'17, April 22-26, 2017, L'Aquila, Italy

DOI: http://dx.doi.org/10.1145/3030207.3030215

[1]The terms response time and delay will be used interchangeably here although they reflect different viewpoints of the same quantity.

analysis of SLAs. First, for analysis mean arrival rates and service times are input parameters and response times are outputs of the analysis. In SLAs arrival rates and response times are input parameters and service rates have to be derived. However, in the product form case, equations can be inverted such that arrival rates and response times become input parameters to compute the required service capacity [6,7]. If numerical analysis approaches or simulation are used for analysis, it is not clear how service capacities are computed from arrival rates and response times. The second disadvantage of queueing network based analysis is more critical. Product form analysis is based on the computation of mean values of result measures based on mean values of the parameters. The specification of mean values in SLAs is usually not sufficient because deviations from the mean can result in unacceptable delays or overload situations. Thus, SLAs usually contain some kind of upper bound which is unavailable in mean value analysis. Thus, a mean value based analysis approach can only over-provision the system to avoid overload situations but it is unclear how much overprovisioning is really required to meet the SLA. It should be mentioned that this also holds for variants where mean value analysis for product form queueing networks is used and it is assumed that only bounds for the parameters are available [14]. The bounds resulting from such an analysis are bounds for mean values, they do not consider the worst case which is often quantified in an SLA.

In performance analysis a calculus for computing bounds on delays and buffer spaces using bounds on arrival and service rates is known since the pioneering work of Cruz in the early nineties [9,10]. These results are the basis for the network calculus [4] and real time calculus [20]. Like queueing network based analysis approaches both calculi compute response time bounds from the knowledge of arrival and service bounds. Therefore they are not directly usable in the SLA context. However, they provide a solid basis to analyze the worst case behavior of systems which is bounded in SLAs.

The work presented in this paper has its foundations in the theory of SLA calculus [6,7] and an earlier version in [23,24]. SLA calculus uses the theory developed in [9, 10] for the analysis of software architectures based on SLAs. This implies that arrival and response time bounds become input parameters to compute bounds on arrivals, departures and response times of composed services and to compute additionally bounds on the minimal service capacity. Like QN analysis, SLA calculus is a mathematical approach to compute the required results in a fairly efficient way. However, without tool support, the approach will hardly be applied because analysis algorithms have to be implemented and models have to be specified in the mathematical formalism which is quite far from the intuitive understanding of the behavior of a service.

In this paper we try to put SLA calculus to work. We introduce an approach to specify composed systems of services graphically, show how an hierarchical analysis is performed and how results are represented. The whole approach has been implemented in a tool which is also introduced here in some detail. The goal is, like in queueing network tools, to hide the details of the analysis from a user by providing an easy to use graphical interface for model specification and result representation.

The paper is structured as follows. Sect. 2 presents some basics of SLA Calculus, followed by a short introduction of input parameter estimation of our models in Sect. 3. SLA Tool is introduced in Sect. 4. Sect. 5 sketches an example of its use. The paper ends with the conclusions and a short outlook of future extensions.

2. THEORETICAL BASIS

We give a brief idea of the analysis approach for further details we refer to [6]. The approach relies on the pioneering work of Cruz [9, 10] on the computation of worst case delays in computer networks. However, in contrast to the available technique we assume that information about the maximal arrival rate and the maximal response time or delay is available from an SLA. In the following subsection we briefly consider how these quantities can be specified in an SLA or can be derived from measurements.

Basically we consider services that provide some result to a user in response to a user request. For the computation of the result some time, denoted as response time or delay, is needed. We begin with the specification of the arrival stream. Service calls arrive in discrete portions and each service call delivers a portion of load to the service. The size of an arriving service call is measured in some application specific unit like the number of transactions, the number of basic operations to compute a result or the size of a data set to be stored, to mention some examples. Let $A : \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}_{\geq 0}$ be the accumulated load that arrived in the interval $[0, t]$. Obviously, $A(t)$ is non-decreasing. In an SLA an upper bound for $A(t)$ is defined in form of an upper arrival curve $\alpha^U(t)$ such that $A(t) - A(s) \leq \alpha^U(t - s)$ for all $t \geq s \geq 0$. Obviously $\alpha^U(0)$ is an upper bound for the size of one service call and $\alpha^{U+} = \lim_{t\rightarrow\infty} \alpha^U(t)/t$, if it exists, is an upper bound for the average arrival rate. After a service call has been processed, it leaves the service. Let $C(t)$ be a function describing departing processed service calls. Obviously $C(t)$ is also non-decreasing and $A(t) \geq C(t)$ because service calls depart after they arrived. For convenience we sometimes define functions on the real axis, then $A(t) = C(t) = 0$ for $t < 0$.

In an SLA specification upper bounds should be defined by simple functions which can be interpreted in terms of the behavior of a software system. We use non-decreasing piecewise linear functions. A non-decreasing piecewise linear function with L segments is defined by L triplets $[x_i, y_i, s_i]$ $(i = 1, \ldots, L)$ such that $x_1 = 0$, $y_1 > 0$, $s_i \geq 0$, $x_i \leq x_{i+1}$, and $y_i + (x_{i+1} - x_i)s_i \leq y_{i+1}$. The function is continuous if $y_i + (x_{i+1} - x_i)s_i = y_{i+1}$. A continuous function is concave if $s_i \geq s_{i+1}$ and convex if $s_i \leq s_{i+1}$. The x_i specify time points, y_i the load that may have entered the system up to time x_i and s_i the slope of the load curve between x_i and x_{i+1} (i.e., the arrival rate). Thus, y_1 is the maximal size of system calls and s_L is an upper bound for the average arrival rate of the load. An example for an upper arrival curve with two segments is shown on the left side of Fig. 4.

By using more than one segment (i.e., $L > 1$) it is possible to specify arrival rates that decrease over time. Usually the arrival process is described by a concave piecewise linear function. y_1 is the amount of load which can arrive immediately and s_1 is the short term arrival rate until time x_2. In contrast to QN analysis where the arrival rate is constant, SLA calculus allows one to consider short term fluctuations in the load.

In a similar way we can define a bound for the response

time or delay induced by arriving service calls. We define a function $F : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ that describes the accumulated delay of service calls of size x. Obviously $F(x)$ is non-decreasing because larger calls require more effort. We define an upper delay curve $\Phi^U(x)$ such that $F(x) - F(y) \leq \Phi^U(x-y)$ for all $x \geq y \geq 0$. Again piecewise linear functions are used for the specification. For example in a system where service calls have a maximal size of 1 and the first two service calls are allowed to have a delay which is twice their size and afterwards the delay is proportional to the size of the call, $\Phi^U = \{[0,0,2],[2,4,1]\}$ would be an appropriate upper bound. The derivative of the delay bound is given by $\phi^U(x) = \left(\Phi^U\right)'(x)$. ϕ^U is piecewise constant if Φ^U is piecewise linear. An example for an upper delay curve with two segments is shown in the middle of Fig. 4.

Observe that the delay bound is a function from the load into the time domain whereas the arrival bound is a function from the time in the load domain. To combine both functions we use the pseudo-inverse of a non-decreasing function $f(t)$.

$$f^{-1}(x) = \inf\{t | f(t) \geq x\} \tag{1}$$

The pseudo-inverse can be easily computed for non-decreasing piecewise linear functions. For a strictly increasing function, the pseudo inverse corresponds to the inverse function. It should be mentioned that in SLA calculus sometimes an inverse has to be computed for more general functions such that the definition of the inverse has to be slightly extended for cases where the functions are not necessarily increasing [6,7]. We will not consider these details here and assume that f^{-1} can be computed in some way.

Additionally, we need the following operations for functions which are all available in the basic approach of Cruz [9].

$$\begin{array}{lcl} (f+g)(t) & = & f(t)+g(t) \\ (f \vee g)(t) & = & f(t) \vee g(t) \\ (f \wedge g)(t) & = & f(t) \wedge g(t) \\ (f \otimes g)(t) & = & \inf_{0 \leq s \leq t} (f(t-s)+g(s)) \\ (f \oslash g)(t) & = & \sup_{0 \leq u} (f(t+u)-g(u)) \end{array} \tag{2}$$

$\vee$ and $\wedge$ are the pointwise maximum and minimum and $\otimes$, $\oslash$ the min/plus convolution and deconvolution [4]. All operations can be easily computed for piecewise linear functions [5].

A user with load requirements that do not exceed α_0^U can immediately use a service with an SLA with arrival and delay bounds (α_1^U, Φ_1^U), if $\alpha_0^U(t) \leq \alpha_1^U(t)$. In this case, the service is analyzed under arrival bound α_0^U and the delay bound remains as defined in the SLA. If $\alpha_0^{U+} = \lim_{t\to\infty} \alpha_0^U(t)/t \leq \alpha_1^{U+} = \lim_{t\to\infty} \alpha_1^U(t)/t$ (for piecewise linear bounds this means $s_{L_0} \leq s_{L_1}$ in our setting), then the user does not overload the service in the long run, which would prohibit the use of the service. However, temporary overload situations may occur such that service calls may have to be delayed to respect the arrival bound of the SLA. An appropriate delay can be introduced by adding a smoother between user arrival process and service. A smoother is known from network calculus [4] and is realized by a leaky bucket or a set of leaky buckets such that the service of the smoother corresponds to α_1^U. An upper bound for the delay in the smoother is then given by

$$\Phi_0^U(x) = \int_0^x \left(\min\left\{\left(\alpha_0^U \oslash \alpha_1^U\right) \otimes \alpha_1^U, \alpha_1^U\right\}\right)^{-1}(y) - \left(\alpha_0^U\right)^{-1}(y) dy \tag{3}$$

An upper bound for the delay of the service and the smoother is $\Phi_1^{U*} = \Phi_0^U + \Phi_1^U$. In the sequel Φ_i^{U*} denotes an upper bound of the accumulated delay of service i resulting from the upper delay bound defined in the SLA plus the delay which is necessary to smooth the arrival process bounded by α_0^U to be consistent with the upper arrival bound α_1^U defined in the SLA. As long as the arrival process of the user is unknown, it is assumed to be equal to α_1^U such that $\Phi_1^{U*} = \Phi_1^U$. If the arrival stream of the user exceeds the capacity of the service, then $\Phi_1^{U*} = \{[0,\infty,\infty]\}$, i.e., the service is overloaded and in the long run response time converges to infinity. In practice this means that additional capacity is required to fulfill the requirements of the user.

The provider has to compute the required service capacity to meet the delay bound under maximal load. Under the reasonable assumption that an increasing arrival rate results in longer delays, a lower bound for the required service rate is given by

$$\sigma_1^L(t) = \left(\left(\phi_1^U + \left(\alpha_1^U\right)^{-1}\right)^{-1} \underline{\oslash} \alpha_1^U\right)(t). \tag{4}$$

The function σ_1^L resulting from piecewise linear functions α_1^U and Φ_1^L is piecewise linear and non-decreasing but not necessarily convex or concave. Thus, it might be substituted by a convex or concave upper bound to specify the minimal service requirements. On the right side of Fig. 4 the lower bound for the service curve resulting from Eq. 4 using the arrival and delay bound on the left and in the middle of the same figure is shown. It can be seen that the resulting service curve is neither concave nor convex.

It is also possible to compute the delay from a known lower bound for the service process using the following relation from [6,7] based on results from [25].

$$\Phi^U(x) = \int_0^x \left(\min\{(\alpha^U \underline{\oslash} \sigma^L) \underline{\otimes} \sigma^L, \sigma^L\}\right)^{-1}(y) - \left(\alpha^U\right)^{-1}(y) dy \tag{5}$$

Equations 3 and 5 are very similar because they both describe the delay in a component with arrival process α and service process σ under FCFS scheduling. In SOAs FCFS scheduling of service calls is usually an appropriate approximation of the real behavior.

For the presented basic relations, results for composed services can be derived. We analyze the maximal delay in a composed system. To compose services we consider three composition operations of two services numbered 1 and 2. The first is the concatenation of services, i.e., after one service the next one is called. Thus, the input process of the second service corresponds to the output process of the first service. Let γ_1^U be the upper bound for the output process of the first service under maximal delay which is given by

$$\begin{array}{l} \gamma_1^U(t+t^*) = \alpha_1^U(t) \\ \text{where } t^* = \sup_t\{t : \forall s \leq t, s + \phi^U(\alpha^U(s)) \geq t\}. \end{array} \tag{6}$$

Since Φ^U and α^U are upper arrival and delay curves, their derivatives are non-increasing. To define an upper bound for the departure process which holds in general, i.e., $C_1(t) - C_1(s) \leq \tilde{\gamma}(t-s)$ for all $t \geq s \geq 0$, the function γ^U has to be shifted by t^* to the left, resulting in

$$\tilde{\gamma}^U(t) = \begin{cases} 0 & \text{for } t < 0, \\ \gamma^U(t+t^*) & \text{for } t \geq 0. \end{cases} \tag{7}$$

Observe that the output process of a service can be more bursty than the input process because the maximal delay of an arriving call may become smaller for calls arriving later when the upper delay bound converges towards the average delay of the system.

The upper delay bound results from the upper bound for the arrival process plus the service calls that are delayed in the service as long as possible. Without knowledge of the concrete input process for the first service, the upper arrival bound for the service is used. Using $\tilde{\gamma}_1^U$ as upper bound for the input process of the second service, the delay and output process of the second service in the sequence can be computed. The delay for load x and of the composed service is

$$\Phi_{12}^{U*}(x) = \Phi_1^{U*}(x) + \Phi_2^{U*}(x). \tag{8}$$

In a parallel fork-join like composition, also denoted as &-composition, two services are called in parallel and the call terminates when both results are available. Thus, both services receive the full input and the upper bounds for delay and output process of the composed service become

$$\Phi_{12}^{U*}(x) = \int_{y=0}^{x} \phi_1^{U*}(y) \vee \phi_2^{U*}(y) dy \text{ and } \gamma_{12}^U(t) = \gamma_1^U(t) \wedge \gamma_2^U(t). \tag{9}$$

The last composition is the parallel choice, also denoted as $\|$-composition, where an arrival stream is distributed among two services. We assume that $p \in (0,1)$ is the portion of the load that is routed to the first service and $\bar{p} = 1 - p$ is the portion that is routed to the second service. Since service calls arrive as discrete portions of load that cannot be split, we have to take into account that complete calls are routed to a service. Let Ω be the maximal size of a call which often equals $\alpha_0^U(0)$ the maximal load that can arrive instantaneously. Then $p\alpha_0^U(t) + \bar{p}\Omega$ and $\bar{p}\alpha_0^U(t) + p\Omega$ are the upper bounds for the arrivals at service 1 and 2, respectively. Using (6) $\gamma_1^U(t)$ and $\gamma_2^U(t)$ can be computed and the upper bounds for delay and output process of the composed service become

$$\begin{aligned} &\Phi_{12}^{U*}(x) = \textstyle\int_{y=0}^{x} \phi_{12}^{U*}(y) dy \\ &\text{where } \phi_{12}^U(x) = \max\left\{\phi_1^U(p(x-\Omega)^+), \phi_2^U(\bar{p}(x-\Omega)^+)\right\} \\ &\text{and } \gamma_{12}^U(t) = \min\left\{\gamma_1^U(t) + \gamma_2^U(t), \alpha_0^U\right\}. \end{aligned} \tag{10}$$

where $(a)^+$ stands for $\max(a, 0)$. It should be mentioned that the behavior of the parallel choice is completely different from a choice in QNs. In a QN is it always advantageous to split load since the delay grows faster than linear with the load. In SLA calculus it is better to route load to one service as long as the service is not overloaded because the delay for the first calls can be longer than the average delay that is achieved after a large number of arrivals. Splitting load and distributing it to more than one service implies then that more calls may have a long delay. A more detailed derivation and interpretation of the bounds can be found in [6, 7].

The bounds can be easily computed for piecewise linear bounds of the arrival and service process. Observe that all compositions result in a composed service with upper bound for the arrival process α_{12}^U, an upper delay bound Φ_{12}^U, and an upper bound for the departure process γ_{12}^U, which can be used in further compositions.

We have presented the whole approach here using upper bounds for arrivals and delays. The computations can be extended by also integrating lower bounds in a similar way (for details see [6, 7]). The availability of additional lower bounds often improves the behavior of a system since it becomes more predictable. Examples will be given below.

3. INPUT PARAMETER ESTIMATION

The set of parameters which is necessary to analyze networks of composed services is limited to upper bounds for arrival rates and delays. In contrast to real time systems or even computer networks the bounds need not be strict, i.e., a small percentage of violations is acceptable. However, this does not necessarily make it easier to specify appropriate bounds. The first and mainly used approach is to use the information available in SLAs. If this is not the case, then bounds have to be estimated ideally based on some data. We will briefly outline both approaches.

Even if SLAs contain information about maximal arrival and delay rates, this information has to be translated into an appropriate bounding curve. The specification of SLAs is not standardized and many different informal and semi-formal approaches to formulate SLAs in different environments exist [13, 22]. Usually an SLA contains a maximal arrival rate of service calls and a maximal size of the calls. If this is the only available information, then curves with one linear segment can be defined. For the response time usually an upper bound plus the percentage of jobs that does not exceed this bound is defined. If only the upper bound is used to define the response time bound, then the model becomes more conservative than the SLA but without knowing the distribution of response times it is impossible to define an upper bound which is met with a given probability. Often the specification of QoS parameters in SLAs is more detailed than just the definition of an upper bound for the mean arrival rate and response time. Most times an interval and an increased rate are defined such that the arrival or delay process can exceed the long time upper bound for some time. Such a specification can be immediately translated in a function with two linear segments. Usually it is sufficient to use functions with only two or three linear segments to capture the information from SLAs. Without a formal specification of SLAs, quantitative information from the SLAs has to be translated manually into appropriate arrival and delay curves.

In some cases, the SLA contains no sufficient information or no SLA is available for a service. Then the bounds have to be estimated. If measurements are available, then these measurements can be used to define an appropriate upper bound. However, in this case some statistical evaluation becomes necessary. We briefly outline the basic approach. Similar ideas have been proposed for real time and network calculus [4, 12].

If $TA(t)$ denotes the load brought into the system by an arrival event at time t, then $A(t) = \sum_{s \le t} TA(s)$. A trace from a measurement contains a finite sequence $TA(s)$ with $s \in [0, T]$ where T is the length of the measurement interval. A tight upper bound is given by

$$\alpha^U(t) = \max_{s \ge 0}\{A(s+t) - A(t)\}. \tag{11}$$

Similar bounds can be derived for delays, if departure events can be linked with corresponding arrival events. The curves for $A(t)$ and $C(t)$ resulting from traces are step functions giving also step functions for the upper bounding curves. However, curves resulting from longer traces will have an

enormous number of segments and are an example for overfitting a model. Thus, we have to find an upper bound with a few linear segments. Let t_i $(i = 1, \ldots, m,\ 0 \leq t_1 \leq t_2 \leq \ldots \leq t_m \leq T)$ be the times when system calls arrive to the system in the interval $[0, T]$ resulting from some measurement taken over the whole interval. a_i is the size of the ith arriving call. Alternatively, we can consider departure times t_i of service calls from the system and d_i is the delay of the departing call. Now assume that $[x_i, y_i, s_i]$ $(i = 1, \ldots, L)$ is a bounding curve. Then the ith arrival (or departure) violates the bounding curve if and only if

$$\exists j < i : \sum_{k=j}^{i} a_k > y_l + s_l(t_i - t_j - x_l) \qquad (12)$$

where $l = \arg\max_{h \in \{1,\ldots,L\}} (x_h \leq t_i - t_j)$. If d_k instead of a_k is used, a condition on delays is defined. For a given bounding curve, the number of arrivals or delays that violate the bound can be computed. Let V be the corresponding number, then V/L is the violation probability.

For a given measurement, a predefined number of segments and a predefined bound of the violation probability, a curve is better than another if the area under the curve is smaller. Thus, the finding of an appropriate curve can be formulated as a linear mixed integer program which is in most cases much too large to be solved exactly such that heuristic optimization algorithms have to be applied. Usually a small percentage of violations should be allowed for arrivals and more important delays since this helps to remove outliers resulting from very long delays. After the bounding curve is available it should be validated against some other measurement of the same system which means to evaluate Eq. 12. If the violation probability for these values is below the required threshold, then the bounding curve can be used. Other more sophisticated techniques for validation can also be applied if sufficient data from measurements is available.

In the following we describe the SLA tool and assume that appropriate bounds for the services have been determined.

4. SLA TOOL

For the analysis of systems using min/plus or max/plus algebra different tools are available. Some software systems support functions in max/plus algebra [8, 18] and can be used as building blocks for system analysis. More appropriate are tools that support modeling in a specific domain. The Real Time Calculus Toolbox [26] is a collection of Matlab functions and Java classes that support system analysis in Real Time Calculus. The specification of the system has to be made at the level of the mathematical functions. DiscoDNC [3] is a Java-based tool for network calculus. Again models are specified at a language level and the mathematical functions have to be used to compute results. The available functions for min/plus analysis cannot be used to perform the analysis of SLAs without significant modifications because it has to be switched between time and load domain several times and as shown in [6,7] the generalized inverse is not sufficient to perform this step in general. To the best of our knowledge all available tools for system analysis based on max/plus algebra are very near to the mathematical basis of the approach which means that the resulting models are very abstract and the tools are hard to use for unexperienced users from the application area.

The goal of SLA tool is to provide a simple interface to the analysis approach which is based on the system model rather than on the mathematical functions. Such an approach is also the base of many queueing network analysis tools like JMT [2] where queueing models are specified graphically or in parameter tables and the approach becomes usable also by unexperienced users. Thus, a Service Oriented Architecture (SOA) is specified graphically as a network of services. Services are connected by the three connection operations, sequential concatenation, parallel ‖-connection and parallel &-connection. In this way an acyclic graph similar to a feedforward queueing network model is generated. Before we introduce the different elements and the composition of models in some more detail, some information about the software structure of the tool will be provided.

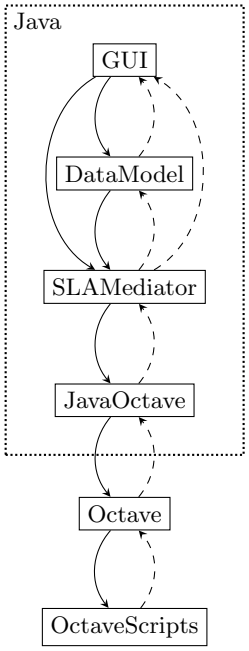


Figure 1: A block diagram showing the tool architecture. Solid lines represent function calls, dashed lines show return of results.

Fig. 1 shows the part of the tool's architecture that is responsible for interacting with Octave. The OctaveScripts block represents a set of Octave functions that implement the mathematical functions as described in Sec. 2. These functions are used for all mathematical operations on curves, such as system analysis or calculating coordinates for plotting curves. As mentioned, curves in our approach are specified by piecewise linear functions and the corresponding objects in Octave are lists of three dimensional vectors $[x, y, s]$ defining a line segment.

JavaOctave provides an interface to Octave for the Java part of the application and allows the execution of strings as commands as well as providing methods to get the results back to Java. For each Octave function exists a corresponding Java method in the SLAMediator that uses parameter objects to generate a command string that can be evaluated by Octave. These commands are forwarded to Octave using JavaOctave. Afterwards the SLAMediator translates the resulting lists back to the Object oriented data model of the tool and returns the final object. The functions pro-

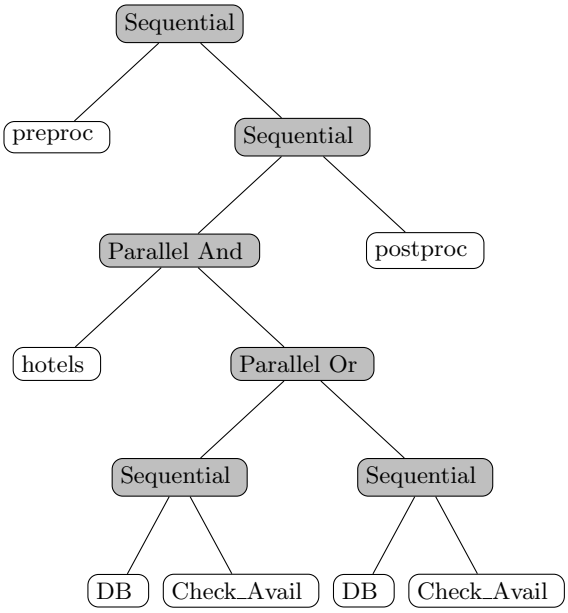


Figure 2: Structure of binary tree for travel agency example (cf. Fig. 6)

vided by the SLAMediator are called from the DataModel and the GUI of the tool. The GUI calls them to plot curves or to check input validity while the DataModel uses these functions to analyze systems. The main difference between the information about the system that is stored in the GUI and the information stored in the DataModel is as follows: While the GUI compositions are directed graphs (as shown in Fig. 5), the data model is a binary tree that can be calculated if the structure of the composition graph is valid. The system analysis uses this tree in order to calculate the resulting curves, which are then returned to the GUI for display. As an example for system analysis we consider the travel agency model of Sect. 5 (cf. Fig. 6). When the tool starts to analyze the composition it generates the corresponding binary tree (cf. Fig. 2). In order to analyze a node of the tree, all its leaves have to be analyzed first. The resulting order of nodes to be analyzed is similar to a depth-first search (see Algorithm 1).

```
<source id="id_0" x="10.5" y="110.5">
  <alphaUpper>{[0,0,1]}</alphaUpper>
  <alphaLower>{[0,0,0]}</alphaLower>
  ...
</source>
<sink id="id_1" x="310.5" y="110.5" />
<service id="id_2" x="110.5" y="110.5" name="Service">
  <alphaLower>{[0,0,0]}</alphaLower>
  <alphaUpper>{[0,1,3],[2,7,2],[3,9,1]}</alphaUpper>
  <phiLower>{[0,0,0]}</phiLower>
  <phiUpper>{[0,0,3],[1,3,2],[3,7,1]}</phiUpper>
  <dialog editDelay="true">
  ...
</service>
```

Table 1: XML description of a single service with source and sink

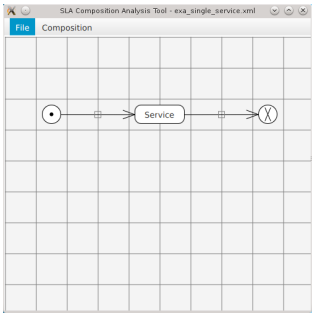


Figure 3: A model of a single service.

The graphical information of a model, the parameters (i.e., the different bounding curves) and additional information is stored in an XML format. Tab. 1 contains the XML specification of the simplest model which can be generated, consisting of a source, a service and a sink. The graphical representation, which is the interface to the user for modeling composed services, is shown in Fig. 3. The XML description includes apart from the model specification, which is shown in Tab. 1, additional graphical information which is not shown here.

For the simple model shown in Fig. 3, the user specifies the model graphically by selecting the basic elements and connecting them. Parameters of a service are the three bounding curves for arrival, delay and service, mentioned above. For the specification of a service, an upper bound for the arrivals and either an upper bound for the delay or a lower bound for the service capacity have to be available. In SLAs upper bounds on arrivals and delays are given, the service capacity is not relevant for a user. However, for a provider it could be interesting to compute a lower bound for the required service capacity. In SLA tool all three curves are associated with a service. Curves are specified by the triplets $[x_i, y_i, s_i]$ and are graphically represented as shown in Fig. 4. A triple of curves belongs to each service. The user can decide which curves are actually shown. Furthermore, the representation of a curve can be modified by choosing a color, the zoom level and the translation of the x and y axis. The graphical representations helps a user to see whether his or her curve fulfills some basic properties like being continuous, concave or convex. The curves belonging to a service are always kept consistent. If upper arrival and delay curves are defined, a lower service curve is implicitly determined and presented. One can switch to the description of the lower service curve, then the upper delay curve is computed. Apart from the upper arrival and delay curves also lower bounds can be defined. Initially lower bounds are $\{[0,0,0]\}$. Lower bounds are not part of SLAs but for an analysis it is sometimes useful to define a lower delay bound to make a service more predictable. From the lower delay bound an upper bound for the service capacity may be computed, although this is usually not needed since service calls that are available too early can be easily delayed, although additional space to store service calls is required in this case. Fig. 4 shows a typical scenario, the upper arrival and delay curve have been specified as concave curves with 2 segments each and the corresponding lower service curve has been computed. Observe that the resulting service curve consists of 4 segments and is neither concave nor convex. In

Figure 4: Bounding curves for arrival, delay and service.

the example the lower arrival and delay curve are $(0, 0, 0)$ resulting in an upper service curve $(0, \infty, \infty)$.

A source has a single parameter, the arrival curve. Again lower and upper bounds can be defined. The parameter of the source specifies bounds for the real arrival rate of the user whereas the parameter of the service describes the bound from the SLA, i.e., the maximal arrival rate the service will accept. A sink has no parameter. If the simple model is analyzed, then the delay and service bounds are computed according to the arrival bounds defined in the source. As long as the upper arrival bound from the source does not exceed the upper arrival bound of the service, the delay bound of the service, as given in the SLA, is valid. However, the lower bound for the service capacity depends on the source arrival bound. If the arrival bound of the source exceeds temporarily the arrival bound of the service, then calls have to be delayed and the upper delay bound grows. The lower bound for the service capacity is in this case the same as computed for the upper arrival bound defined in the SLA. If the mean rate of the source exceeds the mean rate of arrivals for the service, then the delay grows to infinity, as in a queueing network where the arrival rate exceeds the service rate.

We consider an example where
$\alpha_1^U = \{[0, 1, 3], [2, 7, 2], [3, 9, 1]\}$ and
$\Phi_1^U = \{[0, 0, 3], [1, 3, 2], [3, 7, 1]\}$ are the upper bounds for the arrivals and delay of the service. Furthermore, α_0^U is the upper arrival bound of the source. For $\alpha_0^U = \alpha_1^U$, the lower bound for the service process becomes
$\sigma_1^L = \{[0, 0, 0], [1, 0, 3], [3, 7, 2], [4, 9, 1]\}$.
If we choose $\alpha_0^U = \{[0, 1, 1]\}$, then Φ^U remains and
$\sigma_1^L = \{[0, 0, 0], [1, 0, 1], [3, 2, 2], [4, 4, 1]\}$.
For $\alpha_0^U = \{[0, 1, 5], [2, 11, 1]\}$, which temporarily exceeds α_1^U, σ_1^L is the same as for the arrival bound α_1^U and
$\Phi^U = \{[0, 0, 6], [1, 6, 5], [3, 16, 4]\}$ which equals the delay Φ_1^U plus the delay $\{[0, 0, 3]\}$ that is necessary to smooth the arrival stream of the source to be conform to the arrival bound of the service.

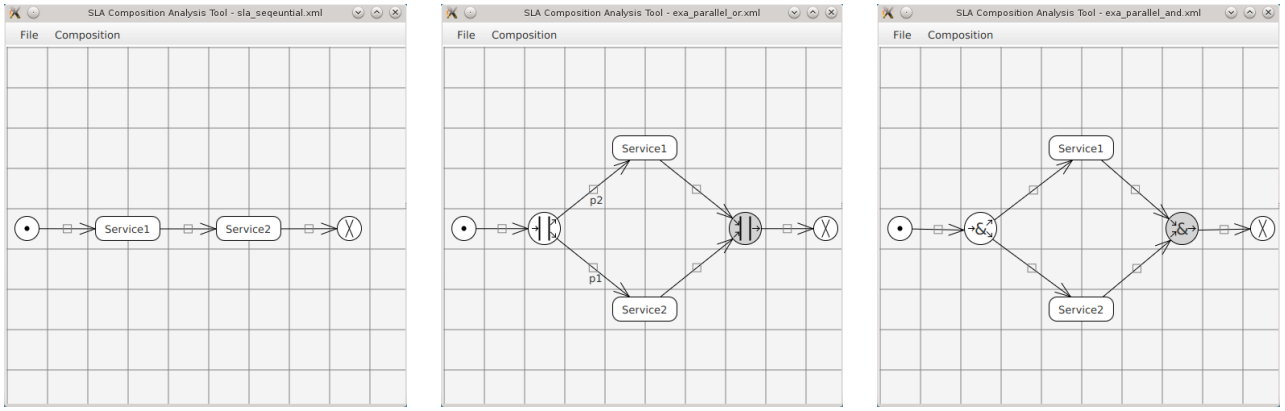
In a SOA a user can combine several services, each with a specific SLA, to build a new composed service which may then be used in further compositions. SLA tool helps to model such compositions and analyze the composed service. Analysis means that delay bounds for composed services are computed, that bounds for the arrival stream into a composed service can be determined and that the required service capacity for the whole composition or a subset of composed services can be computed.

Each composed model describes a flow through a set of services. It starts with a source, where service calls are generated and ends with a sink where processed calls leave the composed service. Basic elements are services and the three composition operators, sequential, &-parallel and ‖-parallel. The graphical representations for specifying the compositions are shown in Fig. 5.

We first describe the sequential composition which can be used for an arbitrary number of services. Fig. 5a shows the composition of two services. We assume that the SLA of the first service is given by α_1^U and Φ_1^U as defined above. For the second service we have $\alpha_2^U = \{[0, 2, 2], [1, 4, 1]\}$ and $\Phi_2^U = \{[0, 0, 2]\}$. For $\alpha_0^U = \alpha_1^U$ the upper delay bound of the composed service becomes
$\Phi_{12}^U = \{[0, 0, 8.667], [1, 8.667, 7.667], [3, 24, 6.667]\}$.
For $\alpha_0^U = \alpha_2^U$ we obtain $\Phi_{12}^U =$
$\{[0, 0, 5.875], [1, 5.875, 4.875], [3, 15.625, 3.875], [4, 19.5, 3.667]\}$.

If the two services are combined by a &-parallel composition, then all calls are split and enter both services. For our example the delay curve equals $\Phi_{12}^U = \{[0, 0, 5]\}$ if arrival curve α_1^U is fed into the composed service and it is
$\Phi_{12}^U = \{[0, 0, 3.2083], [1, 3.2083, 2.2083][3, 7.625, 2]\}$ if α_2^U enters the composed service.

The last composition operator is the ‖-parallel composition which needs as additional parameters the maximal size of a call b and the probability p of choosing service 1. If the two services are composed by a ‖-parallel composition and $p = 0.5$, we obtain for $b = 0$ and arrival process α_1^U $\Phi_{12}^U = \{[0, 0, 3], [2, 6, 2]\}$. A value of $b = 0$ means that ar-



(a) Sequential composition (b) ‖-parallel composition (c) &-parallel composition

Figure 5: Composition of services.

riving load can be split into equal parts, as it is the case for fluid. For $b = 1$ (i.e., calls can have maximal size of 1 and cannot be split), we obtain a slightly different bound $\Phi_{12}^U = \{[0, 0, 3], [2.5, 7.5, 2]\}$. In both cases the delay lies above the delay that would occur if the load is routed to the first service. The reason for this behavior have been described above, the longer delay at the beginning affects in a parallel composition more calls since it occurs in both parallel services. Of course, the delay for a service call converges to 2, the final slope of the delay curves. However, if we increase the arrival stream to $\alpha_0^U = \{[0, 1, 4], [3, 13, 1.5]\}$, then the arrival stream overloads each of the involved services if the complete load is routed to one service. The delay curve becomes
$\Phi_{12}^U = \{[0, 0, 3], [2, 6, 2.52], [22, 56.47, 2]\}$ for $b = 0$, $p = 0.5$ and
$\Phi_{12}^U = \{[0, 0, 3], [2.5, 7.5, 2.88], [26.5, 76.61, 2]\}$ for $b = 1$, $p = 0.5$.

Observe that the parallel composition operators are always used as pairs, an opening operator is followed by a closing operator of the same type. In between can be an arbitrary network of services. To specify more than two parallel services, the parallel operators have to be nested.

A complete model results from composing several subservices. The model is consistent if opening and closing parallel operators fit and the model has a single source and a single sink. Service compositions are always described by acyclic structures which is common in models using min/plus methods for analysis, although there is some work available in generalizing the approach [19]. Here we assume that loops are unwound using several ‖-parallel connectors to define the probabilities for 1, 2, ..., n calls to the service. Of course, such an approach only works for a moderate number of loops.

The analysis of the whole system can be easily realized using a recursive data structure describing a model. Before we introduce the solution algorithm, the recursive data structure is defined. A composition of services in SLA tool is defined by the following grammar.

$$\mathcal{S} = \left(\alpha^L, \alpha^U, \Phi^L, \Phi^U\right) \mid \mathcal{S} \to \mathcal{S} \mid \mathcal{S}\&\mathcal{S} \mid \mathcal{S}\|_{b,p}\mathcal{S} \qquad (13)$$

A service is either a simple service which is specified by the arrival and delay bounds, or it is a composed service. The three composition operators $\to$ for sequential composition, & for &-parallel composition and $\|_{b,p}$ for ‖-parallel composition with parameters b and p combine simple or composed services. A model resulting from the grammar has always a single entry and a single exit point. Models are specified graphically, are mapped implicitly to the proposed grammar that defines a recursive data structure which can be easily generated from the model specification.

In SLA tool the data structure is defined in Octave. Arrival or delay curves are vectors of triples (x, y, s). A simple service is a structure consisting of 4 curves and a composed service is a structure of two services and the composition information (i.e., the composition type and if necessary the parameters b and p). Observe that upper and lower bounds for arrivals and delays are always part of a simple service description. If lower bounds are not specified, they are implicitly set to $\{[0, 0, 0]\}$. In SLA tool simple services can be alternatively characterized by arrival and service capacity bounds. In this case, the delay curves are computed from the bounds for the service capacity and the arrival bounds. Similarly, if service curves are shown in the graphical interface they are always computed from arrival and delay curves. In this way a service is internally always represented by arrival and delay curves and inconsistent specifications are avoided. Computation of a service curve or a delay curve for a service is efficient as long as the number of linear segments is not too large which is usually the case.

The recursive data structure can be naturally used in a recursive algorithm to analyze a composed system. Algorithm 1 analyzes model $\mathcal{S}$ for some arrival stream with upper bound α^U. The algorithm computes upper bounds for the delay Φ and for the departure process γ which might be fed to another service. Again we describe here only the computation of upper bounds. In SLA tool also the computation of lower bounds has been implemented based on [6,7]. Computations follow the recursive structure of the model and simple services are analyzed using the basic equations introduced in Sect. 2 and in some more detail in [6, 7].

Since composed services follow some hierarchical structure, SLA tool supports the building of hierarchical models by composing sub-structures to sub-services. A composition of services with a single entry and exit point can be defined as a sub-service which is a specific node in the graphical representation. Fig. 6 shows an example where the services *flights1*, *flights2* and *hotels* are composed services which in

Algorithm 1 Computation of departure and delay bounds for a composed service $\mathcal{S}$.

```
1: function [Φ, γ] = ANALYZE_SERVICE(α^U, S)
2:     if S == simple then
3:         if α^U ≤ S.α^U then
4:             Φ = S.Φ^U ;
5:         else
6:             Compute Φ_0^U via (3) ;
7:             Φ = Φ_0^U + S.Φ^U ;
8:         end if
9:         Compute γ via (6) ;
10:    else
11:        if S == S_1 → S_2 then
12:            [Φ_1, γ_1] = analyze_service(α^U, S_1) ;
13:            [Φ_2, γ] = analyze_service(γ_1, S_2) ;
14:            Φ = Φ_1 + Φ_2 ;
15:        else
16:            if S == S_1 ‖_{b,p} S_2 then
17:                [Φ_1, γ_1] =
18:                    analyze_service(pα^U + (1 − p)b, S_1) ;
19:                [Φ_2, γ_2] =
20:                    analyze_service((1 − p)α^U + pb, S_2) ;
21:                Φ = Φ_1 ∨ Φ_2 ;
22:                γ = (γ_1 + γ_2) ∧ α^U ;
23:            else /* &-parallel composition */
24:                [Φ_1, γ_1] = analyze_service(α^U, S_1) ;
25:                [Φ_2, γ_2] = analyze_service(α^U, S_2) ;
26:                Φ = Φ_1 ∨ Φ_2 ;
27:                γ = γ_1 ∧ γ_2 ;
28:            end if
29:        end if
30:    end if
31: end function
```

the specific example consist of a simple sequence of two services. The hierarchical structure of the model in terms of sub-services is shown in a separate window. Composed services can be used to reduce the complexity of graphical representations. The analysis is later done using Algorithm 1 on the complete hierarchy.

5. EXAMPLE

As an example we consider an agency which sells travel packages to customers via the Internet. A package consists of a round trip by air and a reservation for a hotel at the destination. The agency's offer is based on the services provided by two flight portals and a hotel reservation system all run by external vendors. The structure of the model is shown in Fig. 6. First a preprocessing step is performed by the agency followed by calls to external services. In order to satisfy customer demands promptly requests for flights and hotels are performed in parallel. For flights two different portals are available that have different types of SLAs. The first flight portal is able to cope with jittery traffic in short time intervals, but accepts less average traffic on the long run than the second flight portal which offers an SLA with shorter long-term delays. Both flight portals offer two services, one returns all flights to the destination inspecting the flight plan via a database request (*DB*) whereas the second service checks the availability of specified flights (*Check_Avail*). The travel agency uses both services to satisfy customer demands. The portal for hotels directly checks for availability of rooms at the destination. Once all requests have been finished, the result information is finalized in a postprocessing step at the travel agency's site.

The SLA specification of all services is given in Table 2. The local services *preproc* and *postproc* as well as the service offered by the portal for hotel rooms seem to be well dimensioned. Considering the flight portals both database requests (*DB*) offer the same SLAs, whereas checking the availability of flights offers different SLAs. At first glance it seems favorable to direct requests to the first flight portal, since it is capable to handle bursty traffic better than *flights2* and guarantees similar response times. On the other hand *flights1* accepts less traffic on the long run, an arrival rate of 5 compared to an average arrival rate of 7 for *flights2*.

If we assume that customer demands are upper bounded by e.g. a bursty arrival curve of $\{[0, 20, 50], [2, 120, 8]\}$ it's obvious that the services of both flight portals need to be used to handle all traffic on the long run and the key question is to determine the portion $p_i, i = 1, 2$ of traffic which should be directed to a flight portal. The long run arrival rate of the source implies $p_1 < \frac{5}{8}$ and $p_2 < \frac{7}{8}$, since the *Check_Avail* services of both flight portals specify the most limiting upper arrival bounds. Considering moderate traffic intensities it seems favorable to use the services of *flights1*, since the upper delay bounds are lower in those situations.

Table 3 shows results of our SLA tool which are here the upper delay bounds that can be guaranteed by the agency directing different portions p_1 of the load to *flights1* by assuming that load can be split arbitrarily according to a fluid model (Note that $p_2 = 1 - p_1$ implying $0.125 < p_1 < 0.625$). It follows from the SLA specifications that it is beneficial to direct as much as possible requests to *flights1*, but only in very low traffic periods. For typical situations it is much better to select $p_1 \approx 0.52$, since the corresponding resultant upper delay bound is optimal. This means that the final de-

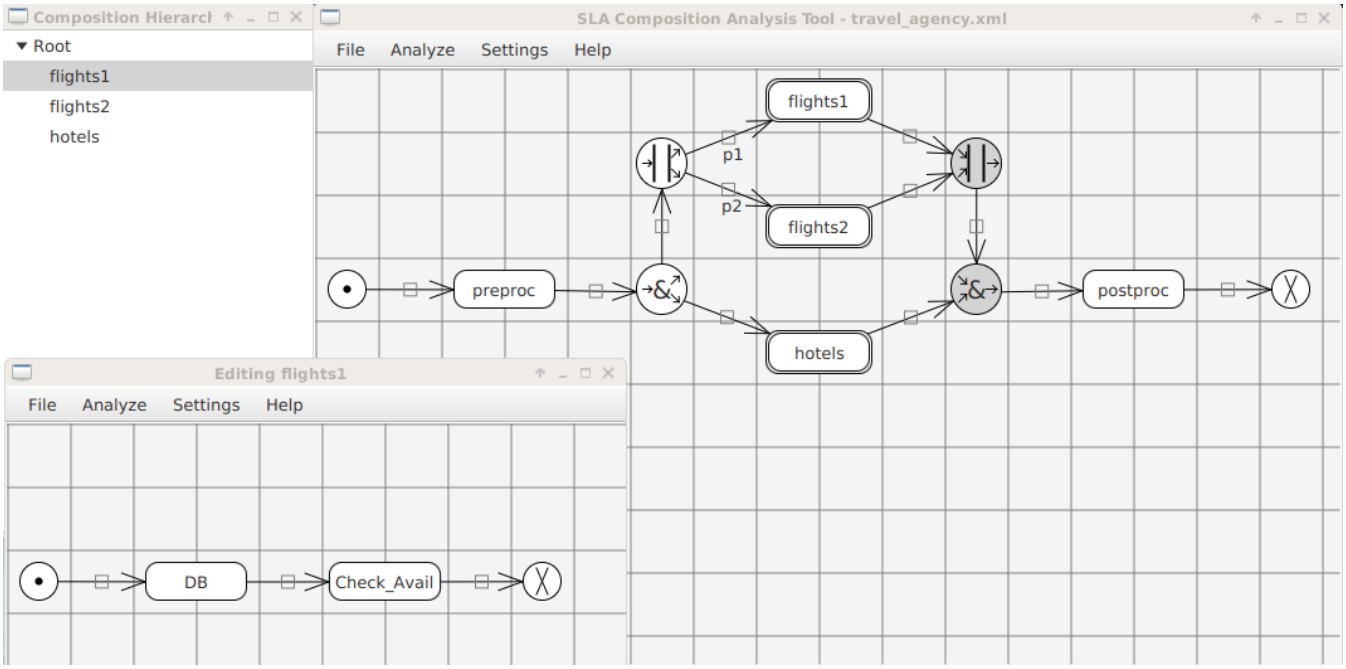


Figure 6: Model of travel agency.

Table 2: Arrival and delay curve specifications for travel agency model

Service	Upper arrival bounds	Upper delay bounds
preproc	$\{[0, 6, 20\}$	$\{[0, 0, 2]\}$
flights1: DB	$\{[0, 6, 12]\}$	$\{[0, 0, 3]\}$
flights1: Check_Avail	$\{[0, 1, 20], [3, 61, 5]\}$	$\{[0, 0, 10], [3, 30, 8]\}$
flights2: DB	$\{[0, 6, 12]\}$	$\{[0, 0, 3]\}$
flights2: Check_Avail	$\{[0, 1, 8], [1, 9, 7]\}$	$\{[0, 0, 20], [1, 20, 6]\}$
hotels	$\{[0, 10, 20]\}$	$\{[0, 0, 3]\}$
postproc	$\{[0, 6, 20]\}$	$\{[0, 0, 2]\}$
Root source	$\{[0, 20, 50], [2, 120, 8]\}$	—

lay of 15 time units per called is reached as soon as possible and the accumulated delay at some given time is minimal.

The situation is very similar if we assume that the load can only be split in portions of size 1. the corresponding results are shown in Table 3. Again $p_1 \approx 0.52$ seems to be the best choice to distribute load among the two services.

6. CONCLUSIONS

This paper presents a new approach to analyze service oriented architectures based on the information available in service level agreements, namely bounds on the arrival stream of service calls and bounds on the response times of a service. These bounds can be adequately described by piecewise linear functions and results for composed services can then be computed using min/plus algebra. In some sense the proposed technique presents an alternative and supplement to the well established queueing network based analysis of software systems.

The proposed analysis techniques have been implemented in SLA tool. Composed services can be specified graphically using basic composition operators such that large models can be easily described and analyzed. The tool is an open source project consisting of a Java based graphical interface and a set of Octave (or Matlab) functions to perform the computations.

The current state of the solution technique and the tool describes a first step. There are several possibilities to extend the approach in future research. Models can be extended using multiple types of service calls resulting in models which are the counterpart to multi-class queueing networks. Furthermore the approach can be combined with optimization techniques to find an optimal service composition.

7. REFERENCES

[1] Danilo Ardagna, Giuliano Casale, Michele Ciavotta, Juan F. Pérez, and Weikun Wang. Quality-of-service in cloud computing: modeling techniques and their applications. *J. Internet Services and Applications*, 5(1), 2014.

[2] Marco Bertoli, Giuliano Casale, and Giuseppe Serazzi. JMT: performance engineering tools for system modeling. *SIGMETRICS Perform. Eval. Rev.*, 36(4):10–15, 2009.

[3] Steffen Bondorf and Jens B. Schmitt. The DiscoDNC v2 - A comprehensive tool for deterministic network calculus. In *8th International Conference on Performance Evaluation Methodologies and Tools, VALUETOOLS 2014, Bratislava, Slovakia, December 9-11, 2014*, 2014.

[4] Jean-Yves Le Boudec and Patrick Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*, volume 2050 of *Lecture Notes in Computer Science*. Springer, 2001.

Table 3: Resultant upper delay bounds of travel agency model for different values of p_1 ($b = 0$)

p_1	Upper delay bound
0.13	$\{[0, 0, 46.96], [1.15, 53.98, 32.96], [169.33, 5597.11, 30.55], [266.67, 8570.9, 28.52], [22156.9, 632804.62, 15]\}$
0.2	$\{[0, 0, 42.55], [1.25, 53.19, 28.55], [169.33, 4852.6, 26.15], [240, 6700.36, 24.92], [1484.17, 37706.85, 15]\}$
0.3	$\{[0, 0, 38.12], [1.43, 54.46, 24.12], [169.33, 4104.93, 21.72], [210, 4988.11, 21.36], [641.43, 14201.82, 15]\}$
0.4	$\{[0, 0, 34.67], [1.67, 57.79, 20.67], [169.33, 3524.21, 18.27], [186.67, 3840.87, 18.27], [374.55, 7272.87, 15]\}$
0.49	$\{[0, 0, 32.32], [1.96, 63.37, 19.46], [6.12, 144.37, 18.32], [169.33, 3134.11, 15.91], [272.6, 4777.34, 15]\}$
0.50	$\{[0, 0, 32.1], [2, 64.2, 19.5], [6, 142.21, 18.1], [169.33, 3098.74, 15.7], [265.33, 4605.47, 15]\}$
0.51	$\{[0, 0, 31.89], [2.04, 65.08, 19.64], [5.88, 140.52, 17.89], [169.33, 3064.51, 15.48], [258.44, 4444.17, 15]\}$
0.52	$\mathbf{\{[0, 0, 31.68], [2.08, 66, 19.83], [5.77, 139.1, 17.83], [169.33, 3055.69, 15.43], [252.38, 4336.73, 15]\}}$
0.53	$\{[0, 0, 31.48], [2.13, 66.98, 20.08], [5.66, 137.91, 18.08], [169.33, 3096.93, 15.67], [278.95, 4814.88, 15]\}$
0.54	$\{[0, 0, 31.28], [2.17, 68.01, 20.32], [5.56, 136.7, 18.32], [169.33, 3136.35, 15.91], [311.76, 5402.33, 15]\}$
0.55	$\{[0, 0, 31.09], [2.22, 69.09, 20.73], [5.45, 136.08, 18.73], [169.33, 3204.78, 16.32], [353.33, 6207.53, 15]\}$
0.56	$\{[0, 0, 30.9], [2.27, 70.23, 21.02], [5.36, 135.08, 19.02], [169.33, 3254.31, 16.62], [407.69, 7214.96, 15]\}$
0.6	$\{[0, 0, 30.2], [2.5, 75.51, 23.58], [5, 134.45, 21.58], [169.33, 3680.11, 19.17], [186.67, 4012.39, 19.17], [1160, 22669.94, 15]\}$
0.62	$\{[0, 0, 29.88], [2.63, 78.64, 26.11], [4.84, 136.27, 24.11], [169.33, 4101.93, 21.7], [190.91, 4570.17, 21.67], [7039.25, 152951.61, 15]\}$

Table 4: Resultant upper delay bounds of travel agency model for different values of p_1 ($b = 1$)

p_1	Upper delay bound
0.13	$\{[0, 0, 47.0], [2.02, 94.92, 33.0], [169.33, 5616.83, 30.6], [267.89, 8632.46, 28.55], [22210.07, 635153.81, 15]\}$
0.2	$\{[0, 0, 42.61], [2.05, 87.36, 28.61], [169.33, 4873.87, 26.21], [241.34, 6760.85, 24.97], [1490.8, 37960.64, 15]\}$
0.3	$\{[0, 0, 38.2], [2.13, 81.3, 24.2], [169.33, 4126.9, 21.79], [211.5, 5045.76, 21.42], [646.41, 14360.53, 15]\}$
0.4	$\{[0, 0, 34.79], [2.27, 78.85, 20.79], [169.33, 3551.99, 18.38], [188.38, 3902.08, 18.38], [381.51, 7451.96, 15]\}$
0.49	$\{[0, 0, 32.38], [2.47, 80.02, 19.48], [6.61, 160.7, 18.38], [169.33, 3152.32, 15.98], [275.42, 4847.4, 15]\}$
0.50	$\{[0, 0, 32.17], [2.5, 80.42, 19.53], [6.5, 158.55, 18.17], [169.33, 3117.12, 15.76], [268.17, 4675.04, 15]\}$
0.51	$\{[0, 0, 31.96], [2.53, 80.88, 19.67], [6.39, 156.83, 17.96], [169.33, 3083.02, 15.55], [261.3, 4513.28, 15]\}$
0.52	$\mathbf{\{[0, 0, 31.75], [2.56, 81.39, 19.88], [6.29, 155.48, 17.88], [169.33, 3071.25, 15.48], [258.4, 4449.68, 15]\}}$
0.53	$\{[0, 0, 31.55], [2.6, 81.96, 20.14], [6.19, 154.31, 18.14], [169.33, 3113.52, 15.73], [285.31, 4938.16, 15]\}$
0.54	$\{[0, 0, 31.36], [2.63, 82.59, 20.44], [6.1, 153.36, 18.44], [169.33, 3164.09, 16.04], [318.56, 5557.48, 15]\}$
0.55	$\{[0, 0, 31.17], [2.67, 83.28, 20.71], [6.0, 152.29, 18.71], [169.33, 3207.9, 16.30], [360.7, 6327.63, 15]\}$
0.56	$\{[0, 0, 31.0], [2.71, 84.04, 21.19], [5.92, 151.95, 19.19], [169.33, 3288.6, 16.79], [415.81, 7426.42, 15]\}$
0.6	$\{[0, 0, 30.29], [2.9, 87.83, 23.8], [5.6, 152.1, 21.8], [169.33, 3722.17, 19.4], [188.38, 4091.6, 19.4], [1210.6, 23918.52, 15]\}$
0.62	$\{[0, 0, 29.97], [3.01, 90.26, 26.35], [5.46, 154.73, 24.35], [169.33, 4144.76, 21.94], [192.57, 4654.66, 21.89], [7269.71, 159589.96, 15]\}$

[5] Anne Bouillard and Eric Thierry. An algorithmic toolbox for network calculus. *Discrete Event Dynamic Systems*, 18(1):3–49, 2008.

[6] Peter Buchholz and Sebastian Vastag. An introduction to SLA calculus for the analytical validation of SLAs. Technical report, Informatik 4, TU Dortmund, 2015. http://ls4-www.cs.tu-dortmund.de/download/buchholz/sla.pdf.

[7] Peter Buchholz and Sebastian Vastag. Towards an analytical method for SLA validation. submitted for publication, 2016.

[8] G. Cohen, S. Gaubert, and J.-P. Quadrat. Maxplus algebra in Scilab and applications. In *Scilab Workshop*, 2001.

[9] Rene L. Cruz. A calculus for network delay, part I: network elements in isolation. *IEEE Transactions on Information Theory*, 37(1):114–131, 1991.

[10] Rene L. Cruz. A calculus for network delay, part II: network analysis. *IEEE Transactions on Information Theory*, 37(1):132–141, 1991.

[11] Heiko Koziolek. Performance evaluation of component-based software systems: A survey. *Perform. Eval.*, 67(8):634–658, 2010.

[12] S. Kunzli, F. Poletti, L. Benini, and L. Thiele. Combining simulation and formal methods for system-level performance analysis. In *Proceedings of the Design Automation Test in Europe Conference*, volume 1, pages 1–6, March 2006.

[13] Dimosthenis Kyriazis, editor. *Cloud Computing Service Level Agreements*. European Commission, 2013.

[14] Johannes Lüthi and Günter Haring. Mean value analysis for queueing network models with intervals as input parameters. *Perform. Eval.*, 32(3):185–215, 1998.

[15] Daniel A. Menascé. Composing Web services: A QoS view. *IEEE Internet Computing*, 8(6):88–90, 2004.

[16] Daniel A. Menascé. Mapping service-level agreements in distributed applications. *IEEE Internet Computing*, 8(5):100–102, 2004.

[17] Daniel A. Menasce and Virgilio A. F. Almeida. *Capacity Planning for Web Services: metrics, models, and methods*. Prentice Hall, 2001.

[18] Jaroslaw Stanczyk, Eckart Mayer, and Jörg Raisch. Modelling and performance evaluation of DES - A max-plus algebra toolbox for Matlab. In *ICINCO 2004, Proceedings of the First International Conference on Informatics in Control, Automation and Robotics, Setúbal, Portugal, August 25-28, 2004*, pages 270–275, 2004.

[19] David Starobinski, Mark G. Karpovsky, and Lev Zakrevski. Application of network calculus to general topologies using turn-prohibition. *IEEE/ACM Trans. Netw.*, 11(3):411–421, 2003.

[20] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. Real-time calculus for scheduling hard real-time systems. In *Proceedings. ISCAS 2000*, pages 100–104, 2000.

[21] Mirco Tribastone, Philip Mayer, and Martin Wirsing. Performance prediction of service-oriented systems with layered queueing networks. In *Leveraging Applications of Formal Methods, Verification, and Validation - 4th International Symposium on Leveraging Applications, ISoLA 2010, Heraklion, Crete, Greece, October 18-21, 2010, Proceedings, Part II*, pages 51–65, 2010.

[22] Jos J. M. Trienekens, Jacques J. Bouman, and Mark van der Zwan. Specification of service level agreements: Problems, principles and practices. *Software Quality Journal*, 12(1):43–57, 2004.

[23] Sebastian Vastag. A calculus for SLA delay properties. In *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance - 16th International GI/ITG Conference, MMB & DFT 2012, Kaiserslautern, Germany, March 19-21, 2012. Proceedings*, pages 76–90, 2012.

[24] Sebastian Vastag. *SLA Calculus*. PhD thesis, Department of Computer Science, TU Dortmund, 2014.

[25] Ernesto Wandeler. *Modular Performance Analysis and Interface-Based Design for Embedded Real-TimeSystems*. PhD thesis, ETH Zürich, 2006.

[26] Ernesto Wandeler and Lothar Thiele. Real-Time Calculus (RTC) Toolbox. `http://www.mpa.ethz.ch/Rtctoolbox`, 2006.