Using Fuzzy Logic to Relax Constraints in GA-Based Service Composition

Massimiliano Di Penta and Luigi Troiano
dipenta@unisannio.it, troiano@unisannio.it
RCOST - Research Centre on Software Technology
University of Sannio, Department of Engineering
Palazzo ex Poste, Via Traiano 82100 Benevento, Italy

ABSTRACT

The rapid diffusion of web services is changing the software engineering landscape. One of the most interesting features offered by service–oriented systems is the possibility to perform dynamic binding, i.e. choosing, among sets of semantically equivalent services, those which better contribute to meet some constraints (e.g., related to the cost or to any other Quality of Service attributes) and optimize some other criteria (e.g., the response time). Solving this problem is NP–hard, and approaches to tackle it using Genetic Algorithms have been proposed.

In some cases, especially when it is not possible to find any solution to the aforementioned problem, it would be useful to relax constraints, in order to find some alternative solutions that, while not meeting the initial constraints, at least offer a reasonable Quality of Service. This paper proposes the use of fuzzy logic to address the imprecision in specifying QoS constraints, estimating QoS values and expressing Service Level Agreements.

Categories and Subject Descriptors

D.2 [**Software Engineering**]

General Terms

Performance, Reliability

Keywords

Service–oriented software engineering, QoS–aware service composition, Fuzzy Logic

1. INTRODUCTION

Web services represent a big novelty in software technology, and are likely to introduce important changes in the software engineering landscape.

GECCO'05, June 25–29, 2005, Washington, DC, USA.
.

As defined by W3C [8], a web service is *a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.*

Mechanisms such as automatic service discovery based on reasoning over service semantic descriptions, or dynamic binding based on Quality of Service (QoS) are just some of the most promising pieces of technology available. In particular, this paper focuses on service late-binding of composite web services, obtained orchestrating more services using glue code [6] or some specific web service orchestration language (e.g., BPEL4WS [1] or WSCI [7]).

Given a specific feature needed in a service orchestration (named *abstract service*), several services (named *concrete services*) realizing such a feature may be available. All concrete services corresponding to an abstract service are functionally equivalent and thus are replaceable by each other. The choice between them can be dictated by non–functional properties, referred to as Quality of Service (QoS) attributes. One may decide to choose the cheapest service, the fastest, or maybe a compromise between the two. On the other hand, the service provider can estimate ranges for the QoS attribute values as part of the contract with potential users (i.e., the Service Level Agreement (SLA)).

Finding a solution of the aforementioned problem, known as QoS–aware service composition, is NP–hard. In the paper [3] authors proposed the use of Genetic Algorithms (GA) for QoS-aware service composition, and showed how, when the number of available services is high, GAs outperforms integer programming, used by Zeng et al. to tackle a similar problem [9].

The main limitation of the proposed approach is that it is able to search only for solutions that meet a given set of constraints, while optimizing the fitness function.

Very often, the following scenarios may arise:

- there is no possible composition able to meet all QoS constraints. Despite that, in some cases the QoS constraint violation may be considered acceptable. For example, we need that our hotel booking composite service is able to complete the transaction within 10 seconds. However, when determining the binding, we find that no combination of concrete services is able to

satisfy such a constraint. As an alternative, we could accept to wait for more than 10 seconds, provided that the response time is bounded within 20 seconds, otherwise this could cause further problems in our process;

- a service integrator may be interested to pursue one of many different QoS tradeoffs. For example, s/he could accept a service able to fulfill its task in less than 3 seconds, and costs less than 10 dollars. As a second choice, s/he could even accept a service that guarantees a response time of 5 seconds, while costing only 7 dollars.

In real word problems, constraints are often imprecisely defined. For instance, if the service costs 10.10 dollars, it is still a good option, but if it costs 20 dollars, it may be unacceptable. Therefore, some constraints may be softly defined. Soft constraints would be related to any QoS attribute, such as time, cost, availability, and so on. Imprecision is not only relatable to QoS attributes, but also to service specification. Sometimes, service characteristics are imprecise. For instance, the response time is assured within a time interval, or reliability would spread around a nominal value. In this paper we address the problem by using Fuzzy Logic, since it has been proved to be a robust technique for dealing with uncertainty, and as it provides the means to consider ranges of values explicitly in the reasoning scheme.

The remainder of this paper is organized as follows. Section 2 overviews the idea of service binding using GA, as also described in the paper [3]. Section 2 introduces the idea of relaxing constraints using fuzzy logic. Finally, Section 4 concludes.

2. SERVICE COMPOSITION USING GENETIC ALGORITHMS

Let us consider a composite service S, composed of n abstract services, $S \equiv \{s_1, s_2, \ldots, s_n\}$. Each component s_i can be bound to one of the m concrete services $cs_{i,1}, \ldots, cs_{i,m}$, which are functionally equivalent. As stated in the introduction and further described in the paper [3], the QoS-aware service composition problem is related to determine a set of concrete services, to be bound to abstract services, so that:

1. QoS (i.e., non functional) constraints, established in the SLA, are met.

2. a function of some other QoS parameters is optimized.

The approach used for computing the QoS of a composite service is similar to what proposed by Cardoso [4]. For a Switch construct in the workflow, each Case statement is annotated with the probability to be chosen. Then, the overall cost of the construct is given by the weighted sum of cost of each Case, where a weight is the Case probabilities. These probabilities are initialized by the workflow designer, and then eventually updated considering the information obtained by monitoring the workflow executions. Loops are annotated with an estimated number of iterations k, and the QoS of the Loop is computed taking into account the factor k. For example, if the Loop compound has a cost C_l, then the estimated cost of the Loop will be $k\ C_l$.

Given a *concretization* of a composite service, i.e., a composite service description where each abstract service has been bound to one of its corresponding concrete services, the overall QoS can be computed by applying the rules described in Table 1, which shows an aggregation function for each pair workflow construct and QoS attribute. The table is not complete and, except that for Loops, the aggregation functions correspond to those proposed by Cardoso [4]. These functions are recursively defined on compound nodes of the workflow. Namely, for a Sequence construct of tasks $\{t_1, \ldots, t_m\}$, the *Time* and *Cost* functions are additive while *Availability* and *Reliability* are multiplicative. The Switch construct of Cases $1, \ldots, n$, with probabilities $p_{a1}, \ldots, p_{an}$ such that $\sum_{i=1}^{n} p_{ai} = 1$, and tasks $\{t_1, \ldots, t_n\}$ respectively, is always evaluated as a sum of the attribute value of each task, times the probability of the Case to which it belongs. The aggregation functions for the fork (named Flow in BPEL4WS) construct, are essentially the same as those for the Sequence construct, except for the *Time* attribute where this is the maximum time of the parallel tasks $\{t_1, \ldots, t_p\}$. Finally, a Loop construct with k iterations of task t is equivalent to a Sequence construct of k copies of t.

2.1 Defining the GA Operators

To let the GA search for a solution of our problem, we first need to encode the problem with a suitable chromosome. In our case, the chromosome is represented by an integer array with a number of items equals to the number of distinct abstract services composing our service. Each item, in turn, contains an index to the array of the concrete services matching that abstract service. Figure 1 gives a better idea of how the chromosome is made.

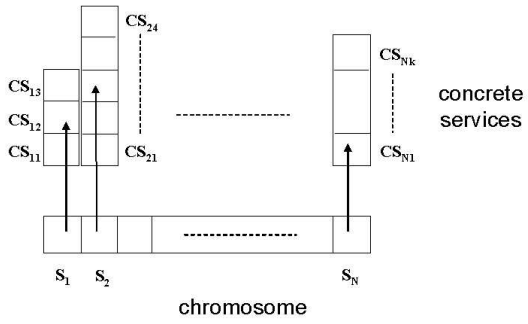

Figure 1: Chromosome Encoding

The crossover operator is the standard two-points crossover, while the mutation operator randomly selects an abstract service (i.e., a position in the chromosome) and randomly replaces the corresponding concrete service with another one among those available. Abstract services for which only one concrete service is available are taken out from the GA evolution.

QoS Attr.	Sequence	Switch	Flow	Loop
Time (T)	$\sum_{i=1}^{m} T(t_i)$	$\sum_{i=1}^{n} p_{ai} * T(t_i)$	$Max\{T(t_i)_{i\in\{1\dots p\}}\}$	$k * T(t)$
Cost (C)	$\sum_{i=1}^{m} C(t_i)$	$\sum_{i=1}^{n} p_{ai} * C(t_i)$	$\sum_{i=1}^{p} C(t_i)$	$k * C(t)$
Availability (A)	$\prod_{i=1}^{m} A(t_i)$	$\sum_{i=1}^{n} p_{ai} * A(t_i)$	$\prod_{i=1}^{p} A(t_i)$	$A(t)^k$
Reliability (R)	$\prod_{i=1}^{m} R(t_i)$	$\sum_{i=1}^{n} p_{ai} * R(t_i)$	$\prod_{i=1}^{p} R(t_i)$	$R(t)^k$
Custom Attr. (F)	$f_S(F(t_i))$ $i \in \{1\dots m\}$	$f_B((p_{ai}, F(t_i)))$ $i \in \{1\dots n\}$	$f_F(F(t_i))$ $i \in \{1\dots p\}$	$f_L(k, F(t))$

Table 1: Aggregation functions per workflow construct and QoS attribute

2.2 Defining the Fitness Function

Once defined the chromosome and the GA operators, our problem can now be modeled by means of a fitness function and, eventually, some constraints. The fitness function needs to maximize some QoS attributes (e.g., reliability), while minimizing others (e.g., cost). When user–defined, domain–specific QoS attributes are used, the specification of the fitness function is left to the workflow designer.

In addition, the fitness function must penalize individuals that do not meet the constraints and drive the evolution towards constraint satisfaction.

A solution x is feasible if it meets constraints on QoS attributes such as cost, response time etc. If the solution overpasses the constraint, then it cannot be accepted. Formally, we have the constraints in the form

$$c(x, l) \leq \kappa(l) \tag{1}$$

where $c(x, l)$ is the value, for the solution x, of the QoS attribute l bouded[1] by $\kappa(l)$.

Generally the bound $\kappa(l)$ is sharp, so that any additional QoS value over $\kappa(l)$ is not allowed. In this case, unacceptable solutions can be simply discarded. However, this is equivalent to explicitly include the constraint satisfaction in the fitness function as $r(x)$

$$f(x) = f(u(x), r(x)) \tag{2}$$

where $u(x)$ is the objective of our optimization problem (e.g., high service reliability or low cost). For example, we may have

$$u(x) = reliability(x) \tag{3}$$

or:

$$u(x) = \frac{1}{cost(x)} \tag{4}$$

Since a good solution tends to maximize $u(x)$ *and* meets as much as possible the constraints, a conjunctive function is the better way to model the solution fitness. For instance,

$$f(x) = \min\{u(x), r(x)\} \tag{5}$$

More in general, we can adopt a generic t-norm $T(\cdot, \cdot)$ [2], so that

$$f(x) = T(u(x), r(x)) \tag{6}$$

[1]For simplicity's sake, we only consider upper bounds. Similar formulae can be obtained for lower bounds.

For precise QoS values and bounds, which entail sharp constraints, $r(x)$ is Boolean. If constraints are not satisfied, then $r(x) = 0$. Since 0 is the *annihilator element* of t-norms, so that $T(u, 0) = 0$ for any T and u, then

$$f(x) = 0 \tag{7}$$

If fitness function is zero, the solution x is not selected for the mating pool. Differently, if constraints are satisfied, then $r(x) = 1$. The value 1 is the *neutral element* of t-norms, so that $T(u, 1) = u$, and

$$f(x) = u(x) \tag{8}$$

Therefore, in case of sharp constraints, the choice of T does not play any role, and the effect on the algorithm is equivalent to discard the unfeasible solutions. However, the explicit inclusion of constraints satisfaction in the fitness function, allows to consider soft constraints, which entail $r(x) \in [0, 1]$. In this case, the choice of T plays a role. Indeed, the minimum operator tolerates too much the weakness of the solutions. A stronger t-norm is the product operator, since $u \cdot r \leq min\{u, r\}$ for any value u and r. In this case,

$$f(x) = u(x) \cdot r(x) \tag{9}$$

3. SOFT CONSTRAINTS

The constraint satisfaction given by $r(x)$ is an aggregated measure of the single constraint satisfaction. Also in this case, a good solution is expected to meet all constraints. This leads again to choose a t-norm. In particular, if we choose the product operator, we get

$$r(x) = \prod_{l=1}^{s} r(x, l)^{w(l)} \tag{10}$$

where $r(x, l)$ is the satisfaction level that the solution x gives for the QoS attribute l, and $w(l) \in [0, 1]$ is the constraint criticality (importance). Normalization of weights requires

$$\max(w(l)) = 1 \tag{11}$$

In case of sharp QoS constraints, it is

$$r(x, l) = \begin{cases} 1 \leftarrow c(x, l) \leq \kappa(l) \\ 0 \leftarrow c(x, l) > \kappa(l) \end{cases} \tag{12}$$

then

$$r(x) = \bigwedge_{l=1..s} r(x, l) \tag{13}$$

where $\wedge$ is the Boolean *and* operator, and $r(x)$ is itself Boolean. In this case, weights do not play any role.

3.1 Imprecise Bounds

Very often, QoS constraints are not sharply defined. Therefore, it is necessary to allow some degree of flexibility in order to relax the constraint. Such a degree can be assumed by considering a maximum acceptable bound. The maximum bound specifies which is the maximum QoS value the service provider is allowed to exceed. Clearly, the more the bound is exceeded, the less favorable the solution is. We can assume a decreasing value of constraint satisfaction level $r(x,l)$ such that

$$r(x,l) = \begin{cases} 1 \leftarrow c(x,l) \leq \kappa_{STD}(i,l) \\ 0 \leftarrow c(x,l) > \kappa_{MAX}(i,l) \end{cases} \quad (14)$$

where κ_{STD} is the standard bound, and κ_{MAX} is the maximum bound allowed. Therefore, there is a continuum of possible bounds in $[\kappa_{STD}, \kappa_{MAX}]$. We are interested in finding a function $\rho(c,l)$ that represents a punctual constraint satisfaction level for the QoS attribute l, bounded at c. In case of precise estimation $c(x,l)$, the constraint satisfaction is given by

$$r(x,l) = \rho(c,l) \quad (15)$$

The simplest way to decrease the satisfaction level within the interval $[\kappa_{STD}, \kappa_{MAX}]$, is to use a linear ramp

$$\rho(x,l) = \begin{cases} 1 & \leftarrow c \leq \kappa_{STD}(i,l) \\ 1-\overline{c} & \leftarrow \kappa_{STD} < c \leq \kappa_{MAX} \\ 0 & \leftarrow c > \kappa_{MAX}(i,l) \end{cases} \quad (16)$$

where

$$\overline{c} = \frac{c - \kappa_{STD}}{\kappa_{MAX} - \kappa_{STD}} \quad (17)$$

The interval $[\kappa_{STD}, \kappa_{MAX}]$ is a set of bounds c, each with an assigned degree of truth $T_l(c)$. If $c(x,l) < \kappa_{STD}$ (i.e., the QoS attribute value is bounded by the standard bound), we do not exceed any limit, then $\rho(c(x,l),l) = 1$. For any QoS estimation $c(x,l) \in [\kappa_{STD}, \kappa_{MAX}]$ we overpass all bounds $c < c(x,l)$, thus the satisfaction level is lower, as the set of bounds met is reduced to $[c(x,l), \kappa_{MAX}]$. If we apply the Sugeno-Takagi's inference calculus [5], that is based on linear interpolation, we obtain

$$\rho(c,l) = \frac{\int_c^{\kappa_{MAX}} t_l(\gamma)d\gamma}{\int_{\kappa_{STD}}^{\kappa_{MAX}} t_l(\gamma)d\gamma} = 1 - \frac{\int_{\kappa_{STD}}^{c} t_l(\gamma)d\gamma}{\int_{\kappa_{STD}}^{\kappa_{MAX}} t_l(\gamma)d\gamma} \quad (18)$$

Therefore, we can assume $t_l(c)$ describes the membership function for the *fuzzy QoS bound* $\kappa(l)$. It is immediate to verify that if $t_l(c) \equiv \Pi(\kappa_{STD}, \kappa_{MAX})$ is a window function as depicted in Figure 2, we get the constraint satisfaction described by Eq.(16).

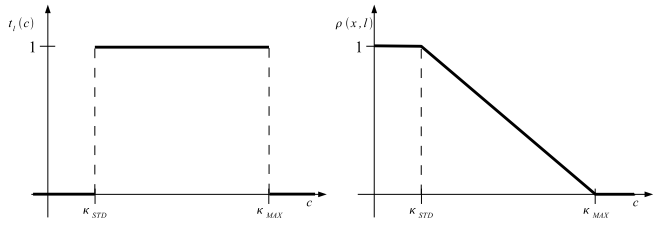

Figure 2: Normal constraint

When the fuzzy bound $\kappa(l)$ is described by a window function, all bounds values have the same degree of truth. We will refer to this condition as *normal constraint elasticity*. The degree of truth $t_l(c)$ is a measure of the bound value c strength: the higher the degree is, the less it is allowed to overpass the standard bound κ_{STD}. Therefore, some constraints might be more restrictive than others, although they consider the same interval $[\kappa_{STD}, \kappa_{MAX}]$. The constraint elasticity (rigidity) is the property of easily (hardly) passing the standard capacity limit κ_{STD}. A *rigid* constraint shows a higher disallowance to overpass the standard limit κ_{STD}, and this disallowance becomes lower when approaching the maximum bound κ_{MAX}. We can characterize a rigid constraint by a linear ramp as depicted in Figure 3.

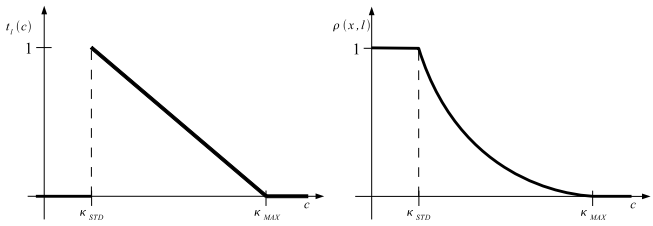

Figure 3: Rigid constraint

The resulting punctual constraint satisfaction (Figure 3) is

$$\rho(c,l) = \begin{cases} 1 & \leftarrow c \leq \kappa_{STD}(i,l) \\ (1-\overline{c})^2 & \leftarrow \kappa_{STD} < c \leq \kappa_{MAX} \\ 0 & \leftarrow c > \kappa_{MAX}(i,l) \end{cases} \quad (19)$$

where $\overline{c}$ is still given by Eq.(17).

Vice versa, an *elastic* constraint allows to overpass the standard bound κ_{STD} more easily, and κ_{MAX} becomes the real maximum bound. We can model this situation as depicted by Figure 4. In this case, the punctual satisfaction

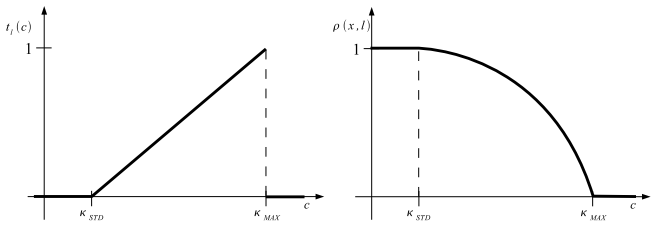

Figure 4: Elastic constraint

level is

$$\rho(c,l) = \begin{cases} 1 & \leftarrow c(x,l) \leq \kappa_{STD}(i,l) \\ 1-\dot{c}(x,l)^2 & \leftarrow \kappa_{STD} < c(x,l) \leq \kappa_{MAX} \\ 0 & \leftarrow c(x,l) > \kappa_{MAX}(i,l) \end{cases} \quad (20)$$

Since the satisfaction level $r(c,l)$ is now within the interval $[0,1]$, the weight associated to the constraint becomes relevant in discriminating criteria that mostly affect the fitness function $f(x)$.

3.2 Imprecise QoS attribute values

Until now we have considered to have precise QoS attribute values available, for which the relation $r(x,l) = \rho(c(x,l),l)$, expressed by Eq.(15) is valid.

However, this is not the case for many QoS attributes. For example, the response time measure is very likely to deviate

from values declared in the SLA or previously monitored (used for service composition purposes).

For simplicity's sake, let us start to consider the simplest case of QoS values estimated by an interval $c(x,l) \equiv [c_1, c_2]$ against a sharp bound $\kappa(l)$. There are three possible cases:

1. $c_2 > \kappa(l)$
2. $c_1 < \kappa(l)$
3. $c1 \leq \kappa(l) \leq c_2$

In the first case, any possible estimation $\gamma \in [c_1, c_2]$ is under the bound $\kappa(l)$, therefore the constraint satisfaction level is $r(x,l) = 1$. Instead, when $c_1 < \kappa(l)$, any possible estimation γ is over the bound, and $r(x,l) = 0$. The last condition describes the case in which part of the possible estimations in $[c_1, c_2]$ is under the bound, and part of them is over. This leads to calculate the constraint satisfaction level as

$$r(x,l) = \frac{\kappa(l) - c_1}{c_2 - c_1} \tag{21}$$

It is immediate to verify that this is a particular application of Sugeno-Takagi integral. More in general, we assume that $c(x,l)$ is modeled by a triangular fuzzy number, as depicted in Figure 5. Values $\gamma \in [c_1, c_2]$ have different degrees

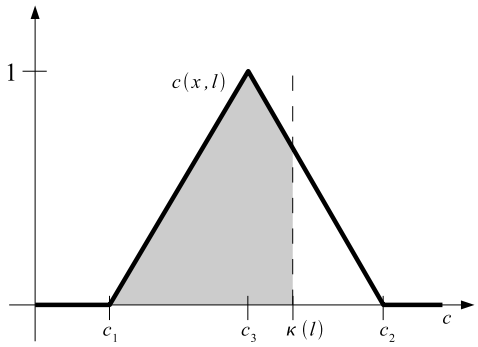

Figure 5: A fuzzy QoS value estimation against a sharp QoS bound

of truth $c(x,l)|\gamma$: some of them are under the bound $\kappa(l)$, then $\rho(\gamma,l) = 1\ \forall\gamma \leq \kappa(l)$; others are over the bound, and $\rho(\gamma,l) = 0\ \forall\gamma > \kappa(l)$. Therefore, the constraint satisfaction can be computed as

$$r(x,l) = \frac{\int_{c1}^{\kappa(l)} c(x,l)|\gamma\, d\gamma}{\int_{c1}^{c2} c(x,l)|\gamma\, d\gamma} \tag{22}$$

This can be geometrically interpreted as the ratio between the triangle area under the bound, and the overall triangle area. When estimation is based on intervals $[c_1, c_2]$, the membership function is $c(x,l)|\gamma = 1\ \forall\gamma \in [c_1, c_2]$, and $c(x,l)|\gamma = 0\ \forall\gamma \notin [c_1, c_2]$. Then, the result of Eq.(22) is Eq.(21).

Now, let us consider the case of interval-based QoS values $c(x,l) \equiv [c_1, c_2]$, against a fuzzy QoS bound $\kappa(l)$. This case is depicted by Figure 6. In this case, the satisfaction level is computed as

$$r(x,l) = \frac{\int_{c1}^{c2} \rho(\gamma,l) d\gamma}{c2 - c1} \tag{23}$$

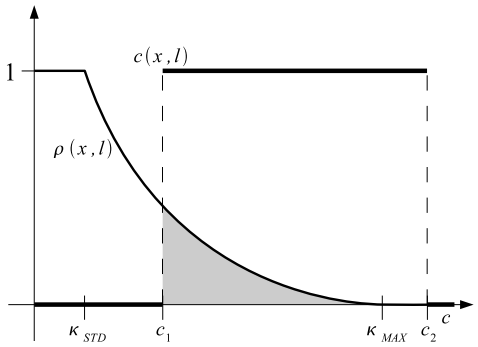

Figure 6: An interval QoS value estimation against a rigid QoS bound

It is easy to verify that in the case of sharp QoS constraint, Eq.(23) provides Eq.(21).

The most general case entails imprecise QoS values against a fuzzy QoS bound, as depicted in Figure 7. The satisfaction

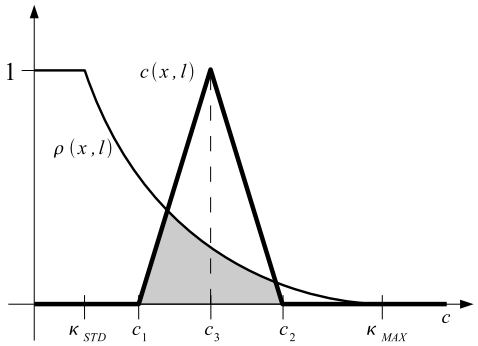

Figure 7: A fuzzy QoS value estimation against a rigid QoS bound

level is provided as

$$r(x,l) = \frac{\int_{c1}^{c2} \min(\rho(\gamma,l), c(x,l)|\gamma) d\gamma}{\int_{c1}^{c2} c(x,l)|\gamma\, d\gamma} \tag{24}$$

When the fuzzy number is an interval, we get Eq.(23); when the constraint is sharp, we get Eq.(22). Since functions are very regular almost everywhere, then integrals can be easily computed with Reimann or Eulero approximation.

4. CONCLUSIONS AND WORK-IN-PROGRESS

QoS–aware composition and binding represents an important challenge for service–oriented software engineering. The paper [3] demonstrates how GAs can be used to determine the set of service concretization that lead to QoS constraint satisfaction and also optimizes an objective function. This paper showed how fuzzy logic can be used to relax QoS constraints and to deal with imprecise QoS value estimations.

This will permit the application of QoS–aware service composition to scenarios where it may not be possible to find any combination of concrete services that meet the constraints, and to properly deal with some QoS attribute, such

as response time, throughput or availability, for which measurements may deviate from previous estimates or from values initially declared from the service provider.

Work in progress is devoted to incorporate this idea in the service composition tool mentioned in the paper [3] and, above all, to perform a thorough experimentation.

5. REFERENCES

[1] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, S. T. D. Smith, I. Trickovic, and S. Weerawarana. Business process execution language for web services. *http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.*

[2] T. Calvo, G. Mayor, R. Mesiar, and A. Adler. *Aggregation Operators: New Trends and Applications (Studies in Fuzziness and Soft Computing, 97)*. Physica Verlag, Reading, 2002.

[3] G. Canfora, M. Di Penta, R. Esposito, and M. Villani. An approach for QoS-aware service composition based on genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, Washington, DC, June 2005 (to appear). ACM.

[4] J. Cardoso. *Quality of Service and Semantic Composition of Workflows*. PhD thesis, Univ. of Georgia, 2002.

[5] M. Sugeno and T. Takagi. *A new approach to design of fuzzy controllers*. Plenum Press, New York, wang, p. p. edition, 1983.

[6] J. Voas, A. Ghosh, G. McGraw, and K. Miller. Glueing together software components: How good is your glue? In *Proceedings of the Pacific Northwest Software Quality Conference*, pages 90–97, Oct 1996.

[7] W3C Working Group. Web Service Choreography Interface. http://www.w3.org/TR/wsci/.

[8] W3C Working Group. Web services architecture. *http://www.w3.org/.*

[9] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web services composition. *IEEE Transactions on Software Engineering*, 30(5), May 2004.