Towards QoS Prediction Based on Composition Structure Analysis and Probabilistic Environment Models

Dragan Ivanović
Universidad Politécnica
de Madrid
idragan@clip.dia.fi.upm.es

Peerachai Kaowichakorn
Universidad Politécnica
de Madrid
p.kaowichakorn@gmail.com

Manuel Carro
Universidad Politécnica de Madrid
and IMDEA Software Institute, Madrid
manuel.carro@imdea.org

***Abstract*—Complex software systems are usually built by composing numerous components, including external services. The quality of service (QoS) is essential for determining the usability of such systems, and depends both on the structure of the composition and on the QoS of its components. Since the QoS of each component is usually determined with uncertainty and varies from one invocation to another, the composite system also exhibits stochastic QoS behavior. We propose an approach for computing probability distributions of the composite system QoS attributes based on known probability distributions of the component QoS attributes and the composition structure. The approach is experimentally evaluated using a prototype analyzer tool and a real-world service-based example, by comparing the predicted probability distributions for the composition QoS with the actual distribution of QoS values from repeated actual executions.**

I. Introduction

Service-oriented [1], [2] and component-based software development have become widely used software construction approaches. The possibility of extensively reusing existing services or components provides higher maintainability and increases cost-efficiency. It also allows developers to design and implement complex systems by combining and extending existing reliable building blocks by means of, for example, service compositions in the form of orchestrations or choreographies.

Aside of external Web services deployed on the Internet, a composition can involve backend application modules, programming language APIs, or third party components. Most of those components are usually seen as black boxes, i.e. the developers know close to nothing about them except for their functional descriptions. The implementation, deployment, environment, location, etc. are all managed by external service providers and are out of reach of service consumers. With such information being unavailable, it is admittedly difficult to analyze the performance of the system to be implemented.

Accordingly, engineering service-oriented systems which respect a given Service Level Agreement (SLA) [3] is hard. Different compositions can implement same functionality, but may, in principle, yield different Quality of Service (QoS) characteristics. Besides, compositions may be able to choose between several services that provide the same functionality, but different QoS. The goal would be to find a composition that fulfills the functional requirements and, at the same time, gives an acceptable (and ideally optimal) level of QoS.

QoS prediction on the composition level is an important and challenging task in the field of service-oriented systems and other complex systems whose boundaries and behavior are not fully specified. Analyzing and predicting composition QoS early during development and pre-production stages can greatly reduce the amount of rework and the cost of maintenance, help the adaptation of software architecture, and increase the overall software quality by exploring design decisions that minimize the risk of violating an SLA. During execution, predicting SLA violations ahead of time can trigger adaptive measures aimed at avoiding the violations or minimizing their undesired effects.

II. Motivation

In a service-oriented system, the composition QoS depends both on the QoS of the individual component services, and on the structure and logic of the composition. This basic model can be further refined by decomposing the QoS observed by the remote client into the effects of the execution environment, such network and processing speeds, and the net (local) QoS behavior of the component services proper.

Stochastic properties of the composition and the component service QoS have been sometimes described using fixed averages or medians. These are neither precise nor informative enough to compute the probability with which an actual QoS value can be expected to fall inside some interval of interest: knowing that some average response time is 3 seconds does not help to know the probability that it answers in 2.5 seconds or less, which is relevant from the point of view of both providers and customers.

An alternative approach is to approximate the ranges of actual QoS values for the components with bounds of admissible values. This is an advantage over the use of constants: it can, at least, give an approximation to the question "what is the probability that the service answers in 2.5 seconds or less". However, in order to retain some degree of completeness (i.e., cover situations that can happen), bounds for the admissible values often need to be enlarged, especially taking into account that services often exhibit a "long-tail" behavior. Having to account for these extreme points reduce the meaningfulness of the observations: bounds express in the end a uniform distribution, and tell us that any value within range has the same probability, which is normally not the case: outliers,

which are are in principle uncommon, will unnecessarily skew the results without contributing to the accuracy of the analysis.

Additionally, and since we are dealing with artificial computational systems whose macro-level behavior can ultimately be traced to that of a discrete system, the actual behavior exhibited is often complex and multi-modal. Therefore we claim that a much higher level of accuracy can be attained by representing and computing the component and composition QoS variability directly with full-fledged probability distributions and operations on them [4].

The challenge here is to obtain the probability distribution of the QoS of compositions from previously known probability distributions of QoS for the component services. Additionally, we want to be able to do that statically and at design time, for which a (faithful) description of the composition is used. In any case, we want the prediction to be obtained without executing the real composition: doing online testing by repeatedly executing the deployed composition and measuring its QoS may be impossible or undesirable for many real-world and mission-critical applications in live systems that involve privacy, security and transactions. Using a precise model for predictions makes it possible, for example, to study the effect of changes before deploying them.

To answer this challenge, the analysis must take into account the structure and logic of the composition. This can always be provided given by a developer (and this is the assumption we made in this piece of work), or, if this is not possible, could in principle be approximately inferred from adequate functional specifications (e.g., interaction protocols, abstract workflows, state machines), process mining [5], etc. We propose an approach that is based on interpretation of the (abstract) composition structure in a probabilistic domain, which takes as input empirically collected QoS distribution profiles for component services, and returns a computed probability distribution for the composition QoS in a single run. The proposed approach maintains a probabilistic description of the composition state variables, together with operations (e.g., integer arithmetic), whose values direct the algorithmic behavior, and thus the overall composition QoS.

This computed probability distribution for the composition QoS can be used to make several predictions. Notably, it can be used to find the probability that QoS corresponding to some execution of the composition will be less than some given quantity (i.e., quoting a previous example, *what is the probability that the service answers in 2.5 seconds or less?*). Note that SLA constraints are often expressed in a similar way.

III. Background And Related Work

A. Quality of Service

In distributed and service-oriented environments, QoS attributes are increasingly important factor in service modeling and selection, and several QoS taxonomies have been proposed [6]–[8]. QoS attributes typically relate to execution time, the required network bandwidth, cost, availability, and other relevant performance and user experience factors. Several standards have been proposed for describing service QoS attributes and reasoning about them [9], [10]. The desirable and acceptable levels of QoS is usually specified in a Service Level Agreement (SLA) [3] between the service provider and consumer. Thus, the analysis of QoS is very important to both service consumers and providers, who have to ensure that QoS of the service does not violate the value given specified in SLA.

B. QoS Analysis

A relatively recent summary of analytic quality assurance (of which QoS analysis is a subset) for Service-Based Systems can be found in [11].

The principles of the classic approaches to analyzing QoS for service compositions have been studied by Cardoso, in a general case [12], and particularly in the context of BPEL processes [13]. That approach is usually known as *QoS aggregation*, i.e., the computation of the aggregate QoS values for the whole composition from the QoS of the individual component services, and was extended by several authors [14]. Most of the QoS aggregation approaches concentrate on the control structure without considering data operations, and express their results as the expected values, which are not sufficient for reasoning about probability of exceeding limits.

Some more recent approaches try to attain a higher level of precision by employing complexity analysis to infer the upper and lower bounds of QoS based on input data and the environmental factors (e.g., the processing and network speeds) [15]. Other proposals [16] use constraint modeling to predict potential and/or imminent SLA violations at runtime. However, both of these approaches operate with the upper and the lower bounds of the component and composition QoS, and thus do not capture the shape of the distribution of values on a finer level of accuracy.

A number of other recent proposals that are concerned more with predicting SLA violations than with computing the composition level QoS (or their distributions), are based on data mining [17], online testing [18], and model checking [19]. We argue that the information that can be derived from computed composition QoS distribution is much richer than a yes/no prediction of some predefined SLA violation.

On the other hand, a recent proposal by Zheng et al. [4] works directly with the QoS statistical distributions, but uses a very abstract composition model that is remote from the practical implementation languages and does not take into account internal composition data and operations, which is bound to give less accurate results.

Another interesting recent proposal by Kattepur et al. [20] uses probability distributions to drive SLA negotiation and optimization for service compositions that are expressed in a *Orc* composition language. While the focus of that approach is complementary to the one of the current paper, it does not take into account the relationship between composition internal state and data operations (including the initial inputs) and its QoS, which is a major concern in the present paper.

IV. METHOD

A. Using Discrete Probability Distributions

We propose a method to represent and compute uncertainty of service composition QoS, the QoS of the component services it invokes, and of its internal state using a uniform framework that operates on probability distributions and relies on interpretation of composition control and data. This approach offers several advantages. Firstly, probability distributions give a much higher level of details than an aggregate central tendency characterization, such as a mean or mode, and do not depend on any assumption of normality, uniformity, etc. of the behavior of the underlying processes. Secondly, empirical observations can be readily converted into the form of probability distributions and fed as inputs to the method. And thirdly, the results from a single analysis of a service composition are equivalent to those that would be obtained from an exhaustive Monte Carlo-style simulation in which the composition is used as a "black box," without taking into account the specifics of its control constructs and data operations.

On the conceptual level, we model service compositions with four main ingredients: the composition structure, the internal composition data (i.e., state), the QoS attribute of interest, and the component services used in the composition.

The composition structure describes the control constructs and data operations. The composition data includes the input given to the composition, as well as any data it uses to drive looping and branching decisions that shape the composition's algorithmic behavior and thus its overall QoS.

The QoS attribute of interest may involve execution time, amount of data sent/received, required bandwidth, number of general or specific operations or component invocations executed, monetary utilization cost, availability, or other performance/quality attributes that can be suitably quantified using some agreed unit of measure. The method is parametric to the choice of the QoS attribute, and only requires (i) the corresponding QoS probability distribution for each component service, and (ii) a suitable aggregation operator for service invocations (see Section IV-E below). For clarity of exposition, in the text that follows we treat a single QoS attribute, but two or more QoS attributes can be analyzed simultaneously and independently of each other.

In the formal model, we use the concept of an *integer random variable* to represent the QoS attribute of interest (such as the execution time) for the composition and each of its component services, as well as the composition's internal state variables. For the QoS attributes, the values are expressed as integral values in some appropriate units of measure, which are precise enough for the particular type of analysis. For the state variables, the domain is either natively discrete (for enumerated and integral types), or is discretized using some appropriate abstraction of data (scalar or structure) size.

The three main groups of the random variables in our composition model, whose typical relationships are shown in Figure 1 are:

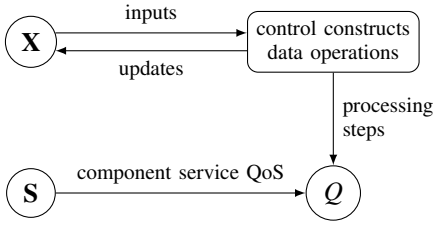


Fig. 1. A typical relationship between the key components.

- $\mathbf{X} = X_1X_2X_3\ldots X_n$ represent the internal composition state variables, including those that represent parts of the incoming and outgoing messages. They typically start from some given input (distribution), and then change depending on the control flow and data operations.
- $\mathbf{S} = S_1S_2S_3\ldots S_m$ represent the QoS attributes for each component service used in the composition. These typically reflect the state of the composition's environment, and their probability distributions come from the empirically collected data.
- Q is the QoS attribute for the composition. It changes in response to both the behavior of the component services $\mathbf{S}$, and to concrete processing steps in the composition, which are interdependent with the state variables $\mathbf{X}$.

Next, we represent the values of the random variables from the set $Q\mathbf{XS}$ of $N = n+m+1$ random variables with a *discrete joint probability distribution*, which is a function:

$$\rho : \mathbb{Z}^N \to [0,1] \tag{1}$$

If $Y_1, Y_2, \ldots, Y_k$ $(0 \le k \le N)$ are distinct variables from $Q\mathbf{XS}$, and $\mathbf{V}$ is an ordered set of $N-k$ remaining variables from $Q\mathbf{XS}$, we write:

$$\rho(Y_1 = y_1, Y_2 = y_2, ..., Y_k = y_k, \mathbf{V} = \mathbf{v}) \tag{2}$$

to denote the probability that $Y_1, Y_2, \ldots, Y_k$ and $\mathbf{V}$ have exactly the values $y_1, y_2, \ldots y_k \in \mathbb{Z}$ and $\mathbf{v} \in \mathbb{Z}^{N-k}$, respectively. When it is clear from the context to what random variables we refer, we write (2) simply as $\rho(y_1, y_2, \ldots, y_k, \mathbf{v})$.

We require that ρ is such that $\sum_{\mathbf{v}\in\mathbb{Z}^N} \rho(\mathbf{v}) = 1$ which means that there exists exactly one true assignment of integer values to the random variables from $Q\mathbf{XS}$. In a singular case, when it is precisely known for some $(q, \mathbf{x}, \mathbf{s}) \in \mathbb{Z}^N$ that $Q = q$, $\mathbf{X} = \mathbf{x}$ and $\mathbf{S} = \mathbf{s}$, then $\rho(q, \mathbf{x}, \mathbf{s}) = 1$, and for all other arguments ρ gives zero. In a non-singular case, we can understand ρ as a lottery from which an actual assignment of values to the random variables is drawn.

In a computer approximation of ρ, the ranges of feasible values for each random variable are always finite, and outside them ρ gives zero. We assume that these ranges are either given at the input to the analysis, or are maintained throughout computations in a manner that is implementation dependent.

B. Probabilistic Interpretation of Compositions

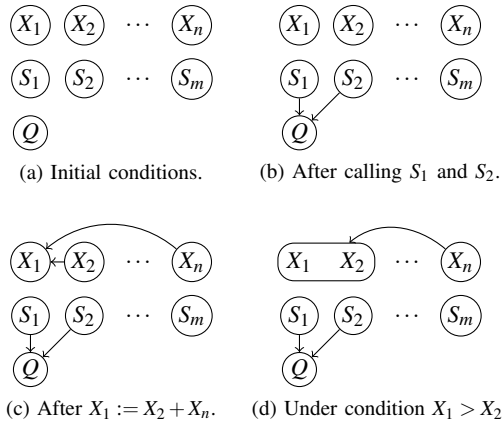
The core of our method is a technique to interpret the control structure and data operations of a given composition in the

$C ::=$ (composition construct)
- $\langle \text{variable} \rangle := E$ (assignment)
- $\mid$ **call** $\langle \text{service} \rangle$ (service invocation)
- $\mid$ **if** B **then** C **else** C (conditional)
- $\mid$ **while** B **do** C (while loop)
- $\mid$ **begin** $C[;\ C]^*$ **end** (sequence)
- $\mid$ **skip** (do nothing)

$E ::=$ (integer expressions)
- $\langle \text{numeral} \rangle \mid \langle \text{variable} \rangle \mid E \circ E \quad (\circ \in \{+, -, *, \textbf{div}, \textbf{mod}\})$

$B ::=$ (Boolean conditions)
- $E \rho E \mid B \wedge B \mid B \vee B \mid \neg B \quad (\rho \in \{>, \geq, =, \neq, <, \leq\})$

Fig. 2. Abstract syntax for composition constructs.



(a) Initial conditions. (b) After calling S_1 and S_2.

(c) After $X_1 := X_2 + X_n$. (d) Under condition $X_1 > X_2$.

Fig. 3. Initial conditions and dependencies.

probabilistic domain, by computing with discrete probability distributions rather than with singular values.

Figure 2 gives a simple abstract syntax for several commonly supported composition constructs, including assignments and integer arithmetic, service invocations, conditionals, loops, and sequential flows.

The interpretation of each construct starts from some ρ before the construct and produces a ρ' after the construct according to the rules (discussed below) that express the semantics of the construct.[1] The interpretation is sound in the sense that all relevant combinations of the random variable values and probabilities from the input distribution are taken into account and represented in the result. This means that the *after* distribution ρ' from a single run of the probabilistic interpreter describes all possible executions that are consistent with the *before* ρ.

C. Initial Conditions and Independence

The interpretation starts from an initial distribution ρ in which all random variables are independent of each other, and each has its own primitive distribution of values, symbolized with unconnected circles in Figure 3(a). For each X_i $(1 \leq i \leq n)$, $\rho_{X_i} : \mathbb{Z} \to [0,1]$ describes the initial value of the state variable,

[1]Note that this is similar to the idea of *before value* and *after value* used to model assignments logically.

x_1	ρ_{X_1}
0	1.0

x_2	ρ_{X_2}
1	0.3
2	0.5
4	0.2

x_n	ρ_{X_n}
0	0.4
1	0.6

x_1	x_2	x_n	$\rho'_{X_1 \mid X_2, X_n}$
1	1	0	1.0
2	1	1	1.0
2	2	0	1.0
3	2	1	1.0
4	4	0	1.0
5	4	1	1.0

x_1	x_2	x_n	ρ'_{X_1, X_2, X_n}
1	1	0	0.12
2	1	1	0.18
2	2	0	0.20
3	2	1	0.30
4	4	0	0.08
5	4	1	0.12

Fig. 4. Sample probabilities for $X_1 := X_2 + X_n$.

which may be singular or represent the distribution of input messages. The aggregate state distribution $\rho_{\mathbf{X}} : \mathbb{Z}^n \to [0,1]$ is computed as:

$$\rho_{\mathbf{X}}(x_1, x_2, \ldots, x_n) = \rho_{X_1}(x_1) \times \rho_{X_2}(x_2) \times \cdots \times \rho_{X_n}(x_n) \quad (3)$$

Next, for each S_j $(1 \leq j \leq m)$, $\rho_{S_j} : \mathbb{Z} \to [0,1]$ represents a distribution obtained by tabulating relative frequencies of the empirically recorded QoS for the corresponding service. The aggregate distribution $\rho_{\mathbf{S}} : \mathbb{Z}^m \to [0,1]$ is computed as:

$$\rho_{\mathbf{S}}(s_1, s_2, \ldots, s_m) = \rho_{S_1}(s_1) \times \rho_{S_2}(s_2) \times \cdots \times \rho_{S_m}(s_m) \quad (4)$$

Finally, $\rho_Q : \mathbb{Z} \to [0,1]$ describes the initial value for the composition QoS, Q. It is normally singular, as the starting execution time, cost, availability of other quality attribute is the part of the QoS attribute definition.

From these components, the overall ρ is computed as:

$$\rho(q, \mathbf{x}, \mathbf{s}) = \rho_Q(q) \times \rho_{\mathbf{X}}(\mathbf{x}) \times \rho_{\mathbf{S}}(\mathbf{s}) \quad (5)$$

This initial case of full independence is modified by service invocations and assignments. For instance, after invoking the component services S_1 and S_2, Q is no longer independent, which is symbolized with arrows in Figure 3(b). In the after ρ' of (5), the independent $\rho_Q(q)$ is replaced with $\rho'_{Q|S_1,S_2}(q \mid s_1, s_2)$ which is the conditional probability of $Q = q$ given $S_1 = s_1$ and $S_2 = s_2$. To simplify the analysis, we assume that the component service QoS distributions do not change and remain independent during the execution of the composition.

D. Assignments and Arithmetic

An assignment has the form $X := E$, where X is from $\mathbf{X}$ and E is an arbitrary expression from Figure 2 which may involve any number of variables from $\mathbf{X}$, including possibly X. The after distribution ρ' needs to satisfy the condition:

$$\rho'(x, \mathbf{v}) = \sum \{\rho(u, \mathbf{v}) \mid x = E[u, \mathbf{v}]\} \quad (6)$$

where $E[u, \mathbf{v}]$ represents the result of E for $X = u$ and $\mathbf{V} = \mathbf{v}$. In (6), the probability for $(x, \mathbf{v})$ in ρ' aggregates the probabilities of all tuples $(u, \mathbf{v})$ form ρ where the expression E evaluates to x, and for all other tuples, ρ' gives zero.

In general, assignments make the variable to the left of ":=" dependent on all those appearing in E. For instance, Figure 3(c) shows the reconfiguration with new dependency

arrows leading from X_2 and X_n to X_1 after (b) and $X_1 := X_2 + X_n$. These new dependencies replace the independent probability $\rho_{X_1}(x_1)$ from (3) with the conditional probability $\rho'_{X_1|X_2,X_n}(x_1 \mid x_2, x_n)$ which tells the probability of $X_1 = x_1$ given $X_2 = x_2$ and $X_n = x_n$.

Computing the conditional distribution $\rho'_{X_1|X_2,X_n}$ is straightforward. Figure 4 shows sample before and after distributions for the same example of assignment (only for feasible values, whose probabilities are greater than zero). The before ρ_{X_1} is discarded. Note that $\rho'_{X_1|X_2,X_n}$ is crisp, because the arithmetic operations are deterministic. The after state distribution ρ'_{X_1,X_2,X_n} is computed as $\rho'_{X_1|X_2,X_n} \times \rho_{X_2} \times \rho_{X_n}$.

Note that the computation of ρ' by introducing the conditional distributions satisfies the condition (6), and its computational cost depends only on the size of feasible intervals of the variables involved in the expression.

An important aspect of the probabilistic assignments and arithmetic is that they can be used for computing both the composition state variables from **X**, and for updating the composition QoS represented by Q from **S** in a unified manner.

E. Service Invocation

A service invocation of the form **call** S_i affects the composition quality Q by composing its previous distribution of values with the random variable S_i that represents the component service's QoS. For cumulative QoS attributes, such as the cost or execution time, which add up from one activity in the execution trace to another, this update can be represented as:

$$Q := Q + S_i \tag{7}$$

which is a special case of the assignment whose effects are computed using equation (6). For other types of QoS attributes, like availability, that represent probability, $+$ in (7) needs to be replaced with $\times$, or, in general, with an appropriate aggregation operator.

If the execution of the invoked service sets some composition state variable X, e.g., one that holds the service result, that can be modeled by replacing the factor that correspond to X in the after $\rho'_{\mathbf{X}}$ with the distribution of the service results.

This approach assumes that both the QoS and the results of the invoked service do not depend on the data that is passed to it as an input message, which is the only assumption we can make without further knowledge of the internal structure of the invoked service and when relying only on the empirically collected data. If, however, we know the structure of the invoked service, we can obtain much more precise results by a probabilistic interpretation of its definition, starting from the known distributions of its inputs, which would give both the distribution for the service quality S_i and the distribution for its outputs, using again the techniques we present in this paper.

F. Sequential Composition

A sequential composition of k constructs has the form:

$$\textbf{begin } C_1;\ C_2;\ \ldots;\ C_k \textbf{ end} \tag{8}$$

where each C_i executes after C_{i-1} (for $1 < i \leq k$). If the interpretation of each construct C_i ($1 \leq i \leq k$) computes after ρ_i from the before ρ_{i-1}, then the whole sequence (8) computes after $\rho' = \rho_k$ from the before $\rho = \rho_0$.

G. Conditionals

We first look at the conditional construct of the form:

$$\textbf{if } B \textbf{ then } C_1 \textbf{ else } C_2 \tag{9}$$

where B is a Boolean condition and C_1, C_2 are the nested constructs. If $\mathbf{v}$ represents the value of all random variables from $Q\mathbf{XS}$, then we can express the probability of executing the **then** part as:

$$p = \sum \{\rho(\mathbf{v}) \mid B[\mathbf{v}]\} \tag{10}$$

where $B[\mathbf{v}]$ represents the truth value of B when $\mathbf{V} = \mathbf{v}$. If $p = 1$ or $p = 0$ we continue simply by interpreting C_1 or C_2, respectively.

If $0 < p < 1$, the initial distributions ρ_1 for C_1 and ρ_2 for C_2 need to reflect the condition B. We have:

$$\rho_1(x, \mathbf{v}) = \begin{cases} 0, & \neg B[x, \mathbf{v}] \\ \rho(x, \mathbf{v})/p, & B[x, \mathbf{v}] \end{cases} \tag{11}$$

and

$$\rho_2(x, \mathbf{v}) = \begin{cases} \rho(x, \mathbf{v})/(1-p), & \neg B[x, \mathbf{v}] \\ 0, & B[x, \mathbf{v}] \end{cases} \tag{12}$$

which ensure the linear decomposition:

$$\rho = p \times \rho_1 + (1-p) \times \rho_2$$

If the probabilistic interpretation of C_1 produces ρ'_1 from ρ_1 and that of C_2 produces ρ'_2 from ρ_2, the result for the whole construct (9) will be their linear combination:

$$\rho' = p \times \rho'_1 + (1-p) \times \rho'_2 \tag{13}$$

To practically separate ρ into ρ_1 and ρ_2, it is sufficient to separate the values of the variables involved in the condition B. Rather than performing a comprehensive filtering of ρ in a straightforward implementation of (10), (11) and (14), we can group and split only the values of the random variables from **X** that appear in B.

Figure 3(d) gives an example of grouping of X_1 and X_2 after (c) and under the condition $B \equiv X_1 > X_2$. First, $\rho_{X_1|X_2,X_n}$ and ρ_{X_2} are replaced by:

$$\rho_{X_1,X_2|X_n} = \rho_{X_1|X_2,X_n} \times \rho_{X_2}$$

which inherits X_1's dependency on X_n, and whose values (continuing the assignment example from Figure 4) are given in the table on top of Figure 5. Next, the table is split into two cases, B and $\neg B$, and the value of p (10) is computed by multiplying with the probabilities of the dependencies, in this case X_n.

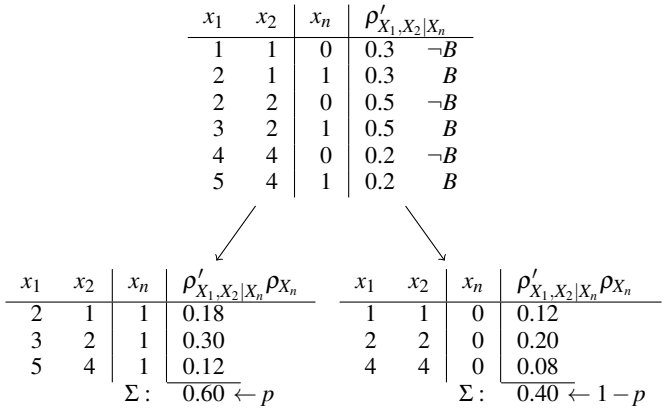


Fig. 5. Grouping and splitting under $X_2 > X_1$.

H. Loops

Upon encountering a loop construct:

$$\textbf{while } B \textbf{ do } C_1 \tag{14}$$

where B is a Boolean condition and C_1 is the loop body, the probabilistic interpreter unfolds it into the conditional:

$$\begin{aligned}&\textbf{if } B \textbf{ then begin} \\ &\quad C_1;\ \textbf{while } B \textbf{ do } C_1 \\ &\textbf{end} \\ &\textbf{else} \\ &\quad \textbf{skip}\end{aligned} \tag{15}$$

which is then executed according to the rules from the previous subsection on conditionals. During the interpretation of the **then** branch, the interpretation will again encounter a copy of (14), which will be again unfolded into a nested conditional with the shape (15). Although in the most general case, this unfolding process may continue indefinitely, we are here interested only in the cases of the terminating service compositions, which are a vast majority of the service compositions in use.[2] We note that, in case of need, there are currently sophisticated termination analyses [21] which can decide automatically whether a given piece of code will terminate or not for generic inputs.

I. Interpreting the Results

We are typically interested in the probability $\Pr[Q \leq a]$ which expresses the probability that the composition quality Q does not exceed some limit $a \in \mathbb{Z}$. This probability can be computed from the resulting ρ' as follows:

$$\Pr[Q \leq a] = \sum_{q \leq a} \sum_{\mathbf{v}} \rho'(q, \mathbf{v}) \tag{16}$$

where $\mathbf{v}$ is a tuple of values for all random variables from **XS**. Note that this means that in practice we are typically not interested in knowing the probability of some value: the question "what is the probability that the process finishes in 3 seconds" is usually uninteresting, and it can even be argued that it tends to zero.

[2]Of course there classes of systems, such as reactive systems, which are not expected to terminate, but these are out of scope for this paper and would likely need a different kind of treatment.

```
?− analyze(if(x>3, call(s1), call(s2)),
           [x=[0−0.2,2−0.4,5−0.4]],
           [s1=[2−0.3,4−0.5,10−0.2],
            s2=[1−0.3,3−0.4,9−0.3]],
           QoS
          ).
QoS = [1−0.18, 2−0.12, 3−0.24, 4−0.2, 9−0.18, 10−0.08]
```

Fig. 6. Sample analyzer invocation.

Other interesting results can be easily derived from the $\Pr[Q \leq a]$, namely $\Pr[Q > a] = 1 - \Pr[Q \leq a]$, $\Pr[Q \geq a] = 1 - \Pr[Q < a]$, and $\Pr[Q < a] = \Pr[Q \leq a - 1]$. Also, if we are interested in knowing the probability that the value of Q lays between some bounds a and b ($a \leq b$), that can be easily computed as:

$$\Pr[a \leq Q \leq b] = \Pr[Q \leq b] - \Pr[Q \leq a - 1]. \tag{17}$$

This is necessary to answer, for example, "what is the probability that the process finishes in 2.95 to 3.05 seconds".

V. Implementation Notes

A fully-functional prototype of the tool has been implemented in Prolog [22]. We have used it to evaluate experimentally the accuracy of our proposal in Section VI.

The main reason for using Prolog is the ease of symbolic representation and manipulation, for both the abstract syntax of compositions and the data on probability distributions. The automatic memory management of Prolog (as in other declarative languages) also helps in quickly producing working prototypes.

The tool receives as input the (abstract) syntax of the composition, a description of the observed QoS for the services, and the expected values of the environment variables. The tool *interprets* the program in a domain of probability distributions and gives as result the expected QoS (time, in our examples). While we are not using it in this paper, the tool also produces the distributions of the internal and output variables of the composition.

The analyzer receives as input the composition definition (in abstract syntax), the initial distributions of values for the state variables, and the QoS distributions for the component services. Its output is the QoS distribution for the composition for the given inputs.

A sample invocation of the analyzer (for the execution time) is shown in Figure 6. The first argument is the representation of the composition structure using Prolog terms, in this case a simple conditional that calls either service S_1 or service S_2 depending on the value of the input message X. The second argument represents the initial distribution for the state variables, here only x. The distribution is specified as a list whose members have the form $i - p_i$, where $i \in \mathbb{Z}$ is a feasible

value, and $p_i \in (0,1]$ is the probability $\Pr[X = i]$, such that $\sum_i p_i = 1$.[3]

The third argument to `analyze` is the list of the execution times for services S_1 and S_2, in some suitable unit of measure. Finally, the fourth argument *QoS* stands for the distribution of the execution times of the composition, which is returned as the result after executing `analyze/4`. The interface includes also calls to read and write data from and to files.

VI. Experimental Validation

In this section we present the design, conduct and results of several experiments whose aim is to evaluate accuracy of QoS the proposed method when used for predicting QoS of a service composition.

A. General Experimental Setup

The experiment is conducted on a fully-deployed service provision/consumption system, and focuses on the execution time as the QoS attribute of interest. The collected execution times corresponding to actual executions of the composition, obtained from a large number of repeated executions, are compared with the predicted distribution of the composition execution time from the single run of the analyzer, to asses the accuracy of the predictions provided by the method.

Services are implemented as Java servlets and deployed on *Google App Engine* [23], a Platform-as-a-Service cloud. The composition is implemented as a client-side Java application that connects to the services, sends the requests, receives the responses, and records the elapsed time. The time is measured and predicted in nanoseconds. The functionality of the composition and the individual services varies from one experiment to another, and is explained below.

The client-side composition code is repeatedly executed to produce the distribution ρ_E for the actual composition execution time, represented by the random variable T_E. The distribution ρ_P of the predicted execution time T_P, is produced from the single run of the analyzer.

The execution time distributions for the services, needed as the input to the analyzer, are the computed from relative frequencies of the empirically collected values from a separate set of several hundreds of individual service invocations.

The composition and the services were written to use all of the structural patterns presented in the previous section in order to validate different combinations of before-to-after distribution computations used in the interpretation method. The algorithms run by the client and the services are given to the analyzer in the abstract syntax representation.

B. Accounting For Network Variability

The execution time analysis needs to take into account the stability of network throughput. Running non-trivial examples 1000 times to obtain the empirical execution time distributions for the component services and the whole composition can take more than 2 hours, and changes in the network conditions during that time can introduce inaccuracies in the result. This is practically unavoidable in a realistic WAN setting, such as the one used in our experimental setting.

[3]Note that "$-$" here is a data constructor, and not a subtraction sign.

```
begin
  x := 0;
  while x < 5 do begin
    call MatrixMultiplicationService;
    x := x + 1
  end
end
```

Fig. 7. Experiment one composition structure.

In order to find out how much network variability affects our results, for each component service we measured time both on the client and the remote end:

1) Total execution time (t_a) is the time observed on the client (i.e., composition) end. It includes the time needed to transmit messages and to execute the component service logic.
2) Net execution time (t_e) is measured on the remote (i.e., component service) end, and is passed to the client side along with the service results. It excludes the time needed to transmit messages.
3) Network transmission time (t_n) is computed as $t_a - t_e$ on the client side.

C. Experiment One: Matrix Multiplication

This experiment aims at testing the accuracy of the prediction in cases involving loops. The deployed remote service performs matrix multiplication. It receives two matrices from the composition, and returns their product. In order to obtain significant execution times, the composition sends large square matrices with dimensions 100×100 or greater. Figure 7 shows the structure of the composition.

To compute the service execution time distributions needed by the analyzer, the multiplication service is called 500 times, and for each invocation the three measurement times (total t_a, net t_e, and network t_n) are recorded.

The composition performs five invocations of the matrix multiplication service, and is invoked 500 times, giving the total of 2.500 component service invocations. The distributions ρ_{Ea}, ρ_{Ee}, and ρ_{En} for the respective actual composition execution (total, net, and network) times T_{Ea}, T_{Ee}, and T_{En} are obtained by measurement, and the analyzer produces the distributions ρ_{Pa}, ρ_{Pe}, and ρ_{Pn} for the respective predicted times T_{Pa}, T_{Pe}, and T_{Pn}..

D. Experiment Two: Sorting

In this experiment tests the accuracy of the prediction of a more complex composition which uses all the patterns mentioned in the previous section.

Two Java servlets are implemented to provide sorting services, one for *BubbleSort* and another one for *QuickSort*. Both services receive an array of integers as input and return a sorted array to the client by using its respective sorting algorithm.

A client-side composition creates 10 arrays of integers between 0 and 99. The size of each array is 1000 elements to ensure significant sorting times for the observation. The client

```
begin
  // Input: value of n from 0 to 9

  // Choosing the sorting mode
  if n < 2 then
    m := 1;
  else if n >= 2 and n < 5 then
    m := 2;
  else if n >= 5 and n < 10 then
    m := 3;

  x := 0;

  // Sort depending on the mode
  if m = 1 then // bubble sort
    while x < 10 do begin
      call BubbleSortService;
      x := x + 1
    end

  else if m = 2 then // quick sort
    while x < 10 do begin
      call QuickSortService;
      x := x + 1
    end

  else if m = 3 then begin // mix sort
    while x < 5 do begin
      call BubbleSortService;
      x := x + 1
    end
    while x < 10 do begin
      call QuickSortService;
      x := x + 1
    end
  end
end
```

Fig. 8. Experiment two: composition structure.

then connects to the servlet as same as in the first experiment to record the total/net execution and the network times.

To generate service time distributions for the analyzer, each service is invoked 500 times, and the relative frequencies of the collected t_a, t_e and t_n values are used to build the corresponding input distributions for the analyzer, which computes the distributions ρ_{Pa}, ρ_{Pe}, and ρ_{Pn} of the respective predicted composition times T_{Pa}, T_{Pe}, and T_{Pn}.

Figure 8 shows the algorithm used by the composition. The input is variable n which determines the sorting mode, which is chosen with uniform probability from the range 0..9. Based on n, the mode m is computed, so that it gets value 1 in 20% of time, value 2 in 30% of time, and value 3 in 50% of time.

The composition is executed 500 times, and the distributions ρ_{Ea}, ρ_{Ee}, and ρ_{En} of the respective actual composition execution times (total, net, and network) T_{Ea}, T_{Ee}, and T_{En}, are recorded, and the distributions of the respective predicted times T_{Pa}, T_{Pe}, and T_{Pn} are again obtained from a single run of the analyzer.

E. Experimental Results

Figures 9, 10, and 11 show the comparison between the predicted probability $\Pr[T_P < t]$ and the actual probability $\Pr[T_E < t]$ for the total, net, and network times, respectively in Experiment One. The corresponding comparisons between

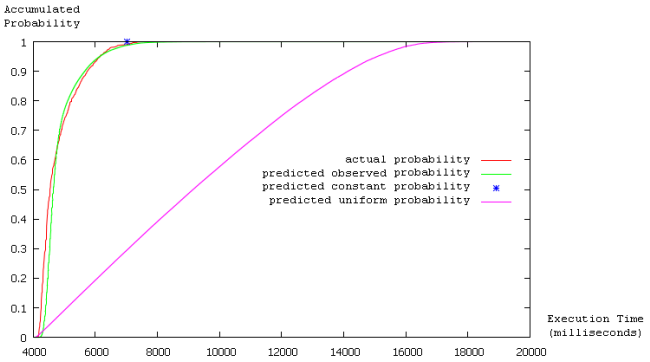


Fig. 9. Experiment 1: Comparison Between $\Pr[T_{Pa} < t]$ and $\Pr[T_{Ea} < t]$

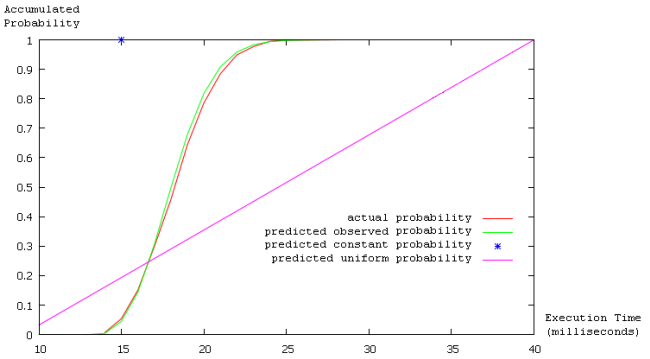


Fig. 10. Experiment 1: Comparison Between $\Pr[T_{Pn} < t]$ and $\Pr[T_{En} < t]$

the predicted and actual probabilities in Experiment Two are shown in Figures 12, 13, and 14. Note that Figures 9 and 11 (resp. Figures 12 and 14) look identical. The reason is that, since they represent total time and network time, the difference between them is net time, which is comparatively small.

The visualization of both experiments suggest the same conclusion. The prediction fits very well with the actual execution. Figures 9 and 12 illustrate that the overall predicted execution time is not perfect. However, Figures 11 and 14 show that most of the errors come from the network transmission time. Additionally, Figures 10 and 13 clearly show that the net execution time prediction (which excludes the network transmission time) has a very small error. According to these figures, it can be concluded that the prediction is very precise.

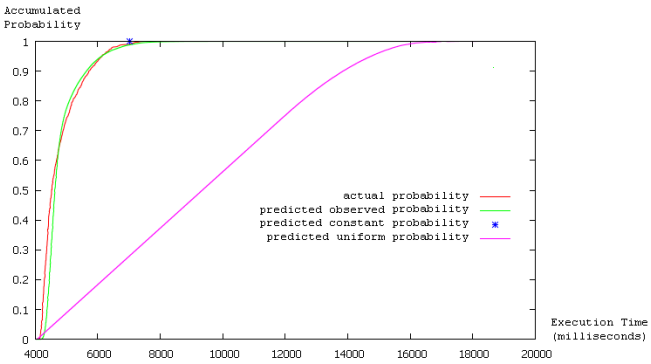


Fig. 11. Experiment 1: Comparison Between $\Pr[T_{Pe} < t]$ and $\Pr[T_{Ee} < t]$

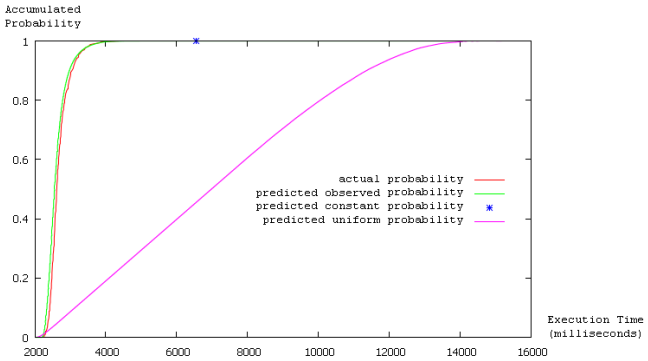


Fig. 12. Experiment 2: Comparison Between $\Pr[T_{Pa} < t]$ and $\Pr[T_{Ea} < t]$

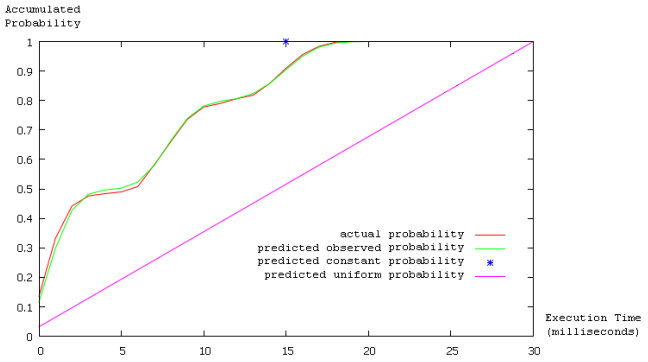


Fig. 13. Experiment 2: Comparison Between $\Pr[T_{Pn} < t]$ and $\Pr[T_{En} < t]$

Note that the cumulative probability graphs shown in the figures are deceptively smooth, due to the great number of data points. In fact, the points in probability distributions are rather noisy, but when added together they exhibit an relatively smooth increase in the cumulative probability.

F. Error Estimation

To further assess the adequacy of our prediction, the *Mean Square Error* (MSE) is calculated in order to evaluate the error of the prediction (Tables I and II), using the following formula:

$$MSE = \frac{1}{t_{\max} - t_{\min}} \sum_{t=t_{\min}}^{t_{\max}} (\Pr[T_P < t] - \Pr[T_E < t])^2 \qquad (18)$$

The smaller the MSE, the more accurate the prediction is. As the MSE gives just a number which is a raw approximation

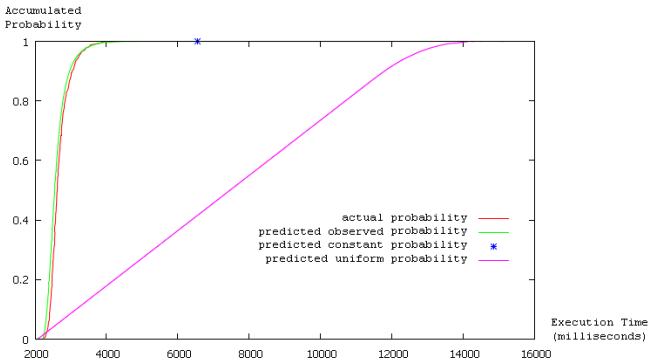


Fig. 14. Experiment 2: Comparison Between $\Pr[T_{Pe} < t]$ and $\Pr[T_{Ee} < t]$

of fitting, it would be desirable to make this comparison against other prediction technique. This is difficult on one hand because it is very time-consuming, and on the other hand because as most papers / studies do not make the source code and all of the details of their benchmarks accessible. Therefore it was decided to take a different path in order to test the accuracy of the approach in this paper against other approaches.

One of the defining characteristics of the approach in this paper is the use of full-fledged probability distributions, while in most other cases predictions are made using just average QoS (i.e., taking a single point as representative of the behavior of some external service) or, at most, upper and lower bounds of QoS (i.e., a uniform distribution between the upper and lower bounds).

Therefore the experiments were repeated by using as input data for the probability distributions either a singleton distribution where a single point (the average) characterizes the behavior of external services (*Constant Probability*) or a uniform distribution ranging from the observed lower to the observed upper bound (*Uniform Probability*). These were generated taking as starting point the observed probability distribution of the services. Since the uniform probability distribution and the constant probability distribution contain less information than the observed probability distribution, we expect the predictions performed using them to be less accurate than those performed with the observed probability distribution. The results are depicted in the graphs as (blue) star for the *constant probability* and as a purple curve for the *uniform probability*, and they are evaluated in *Table I* and *Table II* for the first and the second experiment respectively.

Tables I and II show a comparable landscape: it is clear that using the observed probability distribution produces predictions which are much more accurate (the MSE is smaller) than those generated using either the uniform probability distribution or the constant probability distribution. It can also be seen that most of the prediction errors come from $\Pr[T_{Pn} < t]$, which is difficult to control. If the network issues are excluded ($\Pr[T_{Pe} < t]$), the predictions show very promising results with very small MSE.

Measurement	*Observed Probability*	*Uniform Probability*	*Constant Probability*
$\Pr[T_{Pa} < t]$	0.070	0.383	0.577
$\Pr[T_{Pn} < t]$	0.012	0.310	0.434
$\Pr[T_{Pe} < t]$	0.0003	0.138	0.537

TABLE I
EXPERIMENT 1: MEAN SQUARE ERROR

Measurement	*Observed Probability*	*Uniform Probability*	*Constant Probability*
$\Pr[T_{Pa} < t]$	0.006	0.388	0.494
$\Pr[T_{Pn} < t]$	0.010	0.306	0.383
$\Pr[T_{Pe} < t]$	0.0001	0.126	0.488

TABLE II
EXPERIMENT 2: MEAN SQUARE ERROR

VII. Conclusions and Future Work

We presented an approach to predict the QoS of service-oriented systems by using probability distributions. The prediction is useful for the point of view of both service providers and service consumers in order to make better-informed decisions which involve the quality of service agreements in an SLA. At the moment, the method covers the basic coding patterns common to many programming / coordination languages, and can also be applied to other structured systems which are not implemented through computers (i.e., human-provided services).

We also present and use in the evaluation a working prototype which analyzes the input composition w.r.t. a description of the input arguments and the QoS of the services it accesses and produces the predicted probability distribution of QoS of the program. The experimental evaluation show that our approach is able to produce accurate predictions.

We are working on improving the prototype to perform e.g. reverse calculations (i.e., which probability distributions should characterize the services so that certain qualities in the output are met). We also want to improve the internal algorithms and representations. At the moment, all point-wise calculations are exact, which may be a drawback in large examples. We plan to explore abstraction mechanisms (i.e., transforming probability samplings to reduce the number of points) which trade precision for speed and interpolation mechanisms which avoid representing all points in the domain or which make it possible to answer questions for QoS values for which no data is available.

As in all systems relying on historical, the accuracy of the predictions depends on how up-to-date is the data of the QoS of the external services. As services evolve, their QoS profile can change (due to e.g. hardware / software updates, changes in physical location, network speed, and bandwidth, etc.) These changes are in general unknown to customers, and the data regarding the QoS of the services can be easily get outdated. Hence, QoS data should be updated periodically. Besides being time consuming, this requires a strategy to decide when this data needs to be refreshed and a non-intrusive technique (e.g., online data refreshing) to actually refresh it.

Acknowledgments

The research leading to these results has received funding from the European Unions Seventh Framework Programme (FP7/2007-2013) under grant agreement n 258862 *4Caast*, from the Madrid Regional Government under CM project P2009/TIC/1465 (PROMETIDOS), and from the Spanish Ministry of Economy and Competitiveness under projects TIN-2008-05624 *DOVES* and TIN2011-39391-C04-03 *StrongSoft*. The authors would also like to thank the European Commission Erasmus Mundus programme.

References

[1] M. P. Papazoglou and D. Georgakopoulos, "Service-Oriented Computing," *Communications of ACM*, vol. 46, no. 10, pp. 24–28, 2003.

[2] F. Curbera, B. J. Krämer, and M. P. Papazoglou, Eds., *Service Oriented Computing (SOC)*, ser. Dagstuhl Seminar Proceedings, vol. 05462. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, November 2006.

[3] W. Sun, J. Zhang, and F. Liu, "WS-SLA: A Framework for Web Services Oriented Service Level Agreements," *Computer Supported Cooperative Work in Design, 2006. CSCWD '06. 10th International Conference on*, pp. 1–4, May 2006.

[4] H. Zheng, J. Yang, W. Zhao, and A. Bouguettaya, "QoS Analysis for Web Service Compositions Based on Probabilistic QoS," in *Proceedings of the 9th international conference on Service-Oriented Computing*, ser. ICSOC'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 47–61. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-25535-9_4

[5] W. M. P. van der Aalst, "Process mining," *Commun. ACM*, vol. 55, no. 8, pp. 76–83, 2012.

[6] B. Sabata, S. Chatterjee, M. Davis, J. Sydir, and T. Lawrence, "Taxonomy for QoS specifications," *Third International Workshop on Object-Oriented Real-Time Dependable Systems*, pp. 100–107, 1997.

[7] M. Godse, U. Bellur, and R. Sonar, "A Taxonomy and Classification of Web Service QoS Elements," *Int. J. Commun. Netw. Distrib. Syst.*, vol. 6, no. 2, pp. 118–141, Feb. 2011.

[8] T. O. Group, "Summary of Quality Model for Web Services," The Oasis Group, Tech. Rep., 2005.

[9] C. Zhou, L.-T. Chia, and B.-S. Lee, "Web Services Discovery with DAML-QoS Ontology," *Int. J. Web Service Res.*, vol. 2, no. 2, pp. 43–66, 2005.

[10] S. Bajaj, D. Box, D. Chappell, F. Curbera, G. Daniels, P. Hallam-Baker, M. Hondo, C. Kaler, D. Langworthy, A. Nadalin, N. Nagaratnam, H. Prafullchandra, C. von Riegen, D. Roth, J. Schlimmer, C. Sharp, J. Shewchuk, A. Vedamuthu, c. Ümit Yal and D. Orchard, *Web Services Policy Framework (WS- Policy)*, IBM, March 2006.

[11] A. Metzger, S. Benbernou, M. Carro, M. Driss, G. Kecskemeti, R. Kazhamiakin, K. Kritikos, A. Mocci, E. D. Nitto, B. Wetzstein, and F. Silvestri, "Analytical Quality Assurance," in *S-CUBE Book*, ser. LNCS, M. P. Papazoglou, K. Pohl, M. Parkin, and A. Metzger, Eds., vol. 6500. Springer, 2010, pp. 209–270.

[12] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of Service for Workflows and Web Service Processes," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 3, pp. 281 – 308, 2004.

[13] J. Cardoso, "Complexity Analysis of BPEL Web Processes," *Software Process: Improvement and Practice*, vol. 12, no. 1, pp. 35–49, 2007.

[14] M. Dumas, L. García-Bañuelos, A. Polyvyanyy, Y. Yang, and L. Zhang, "Aggregate Quality of Service Computation for Composite Services," in *ICSOC*, 2010, pp. 213–227.

[15] D. Ivanović, M. Carro, and M. Hermenegildo, "Towards Data-Aware QoS-Driven Adaptation for Service Orchestrations," in *Proceedings of the 2010 IEEE International Conference on Web Services (ICWS 2010), Miami, FL, USA, 5-10 July 2010*. IEEE, 2010, pp. 107–114.

[16] ——, "Constraint-Based Runtime Prediction of SLA Violations in Service Orchestrations," in *Service-Oriented Computing – ICSOC 2011*, ser. LNCS, G. Kappel, H. Motahari, and Z. Maamar, Eds., no. 7084. Springer Verlag, December 2011, pp. 62–76, best paper award.

[17] P. Leitner, W. Hummer, and S. Dustdar, "Cost-Based Optimization of Service Compositions," *IEEE Transactions on Services Computing*, vol. 99, no. PrePrints, 2011.

[18] O. Sammodi, A. Metzger, X. Franch, M. Oriol, J. Marco, and K. Pohl, "Usage-based Online Testing for Proactive Adaptation of Service-based Applications," in *COMPSAC 2011 – The Computed World: Software Beyond the Digital Society*. IEEE Computer Society, 2011.

[19] E. Schmieders and A. Metzger, "Preventing Performance Violations of Service Compositions Using Assumption-Based Run-time Verification," in *ServiceWave 2011*, ser. LNCS, A. Zisman, I. Llorente, S. M., A. W., and V. J., Eds. Springer, 2011.

[20] A. Kattepur, A. Benveniste, and C. Jard, "Negotiation strategies for probabilistic contracts in web services orchestrations," in *ICWS*, 2012, pp. 106–113.

[21] B. Cook, A. Podelski, and A. Rybalchenko, "Proving program termination," *Commun. ACM*, vol. 54, no. 5, pp. 88–98, 2011.

[22] L. Sterling and E. Y. Shapiro, *The Art of Prolog - Advanced Programming Techniques, 2nd Ed.* MIT Press, 1994.

[23] G. D. Team, "Google App Engine," https://developers.google.com/appengine/, last accessed on 31st August 2012.