A Theory of QoS for Web Service Orchestrations

Sidney Rosario, Albert Benveniste, Claude Jard

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Theory of QoS for Web Service Orchestrations

Sidney Rosario — Albert Benveniste — Claude Jard

N° 6951

Juin 2009

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--6951--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

A Theory of QoS for Web Service Orchestrations*

Sidney Rosario†, Albert Benveniste‡, Claude Jard§

Thème COM — Systèmes communicants
Équipes-Projets DistribCom

Rapport de recherche n 6951 — Juin 2009 — 33 pages

Abstract: While extensive foundational work exist for the functional aspects of Web service orchestrations, very little exists regarding the foundations of Service Level Agreements (SLA), Service Level Specifications (SLS), and more generally Quality of service (QoS) issues. In this paper we develop a comprehensive theory of QoS for Web service Orchestrations. To support multi-dimensional or composite QoS parameters, QoS domains must be *partially*, not totally, ordered. We identify the needed algebra to capture how QoS get transformed when synchronising service responses and to represent how a service call contributes to the end-to-end QoS of the orchestration. SLA/SLS approaches implicitly assume that, the better a called service performs, the better the orchestration does. This property, called *monotonicity,* does not always hold, however. We provide conditions ensuring it. Then we show how SLA or contracts between the orchestration and the services it calls can be *composed* to derive an SLA or contract between the orchestration and its clients. To account for high variability in measured QoS parameters for existing Web services, we support both *probabilistic* and non-probabilistic approaches. Finally, we propose a mild extension of the Orc language for service orchestrations to support flexible QoS management according to our theory.

Key-words: services orchestrations, QoS, Petri nets, occurrence nets

* This work was partially funded by the ANR national research program DOTS (ANR-06-SETI-003), DocFlow (ANR-06-MDCA-005) and the project CREATE ActivDoc.
† INRIA-Rennes, IRISA, Campus de Beaulieu, Rennes France
‡ INRIA-Rennes, IRISA, Campus de Beaulieu, Rennes France
§ ENS Cachan, IRISA, Université Européenne de Bretagne, Bruz France

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Une théorie de la Qualité de Service pour les orchestrations

Résumé : Si de nombreux travaux existent concernant les fondements des orchestrations de service en ce qui concerne les aspects fonctionnels, il en va tout autrement pour les aspects de Qualité de Service (QoS) et leurs contrats (SLA/SLS). Cet article propose une théorie globale pour ces sujets. Notre théorie prend en compte la QoS multi-dimensionnelle, et, pour ce faire, nous considérons des domaines de QoS partiellement ordonnés et nous formalisons les opérations algébriques nécessaires à la manipulation de la QoS. Pour être fondée, toute approche de la QoS par contrats suppose une propriété de *monotonie* des orchestrations: si un sous-contractant dépasse la performance promise par contrat, alors la performance de l'orchestration doit en être améliorée. Cette propriété est loin d'être anodine: nous l'étudions en détail dans cet article. Puis nous étudions la *composition* de contrats, c'est-à-dire la synthèse du contrat liant l'orchestration à son client, en fonction des contrats liant l'orchestration à ses sous-contractants. Notre approche s'applique à des contrats de nature probabiliste. Enfin, nous proposons une extension du langage Orc permettant d'appliquer notre théorie à ce langage.

Mots-clés : orchestrations de services, qualité de service, réseaux de Petri, réseaux d'occurrence

Contents

1 Introduction

While extensive foundational work exist for the functional aspects of Web service orchestrations [10, 4], very little exists regarding the foundations of Service Level Agreements (SLA), Service Level Specifications (SLS), and more generally Quality of service (QoS) issues [3, 6, 19].

Measurements on existing Web services reveals that typical QoS parameters such as response time vary a lot from call to call, see [13] for a report of such measurements and related discussion. From this fact, the need emerges for a *probabilistic* approach to QoS, in which SLAs are formulated in probabilistic terms. Such an approach was developed in [12, 13], for the special case of response time. In [14], we proposed an extension of the former study to handle general QoS parameters, possibly composite, in a probabilistic setting.

Central to this approach is the notion of *contract* that properly formalizes SLA/SLS from a mathematical standpoint. A contract formalizes the obligations of a service towards its client, the *guarantees*, and the obligations of the client when calling the service, the *assumptions*. Obligations of the service may involve various QoS parameters such as response time (also called latency), security, quality of response (valid, exception, etc.). The obligations of the client typically concern query rate, but could also involve the complexity of the queries.

In this paper, we provide the foundations for *a mathematical theory of probabilistic contracts, for general QoS aspects,* with the following main contributions:

Competition policy: this policy formalizes how QoS affects the semantics of orchestration when synchronizing responses and when preferring some responses over others, based on QoS considerations.

Monotonicity of orchestrations: An orchestration is monotonic (with respect to QoS) if, when a called service performs better, then so does the orchestration as a consequence. Monotonicity is indeed always implicitly assumed when considering SLAs. It always holds if a pessimistic definition of end-to-end QoS is taken for the orchestration. If, however, a tighter definition is preferred (to avoid excessive pessimism when offering contracts to the orchestration's client), then monotonicity no longer holds in general. Monotonicity was studied in [5] for the restricted case of latency, for both probabilistic and non-probabilistic settings. In this paper we extend this study far beyond this restricted case, by encompassing general QoS parameters.

Probabilistic contract composition: This is the process of deriving a contract which can be offered to the orchestration's client, from contracts established with the services called by the orchestration. This involves both *assumptions* and *guarantees* regarding the QoS. While guarantees offered by the orchestration are naturally derived from the gurarantees its called services offer, assumptions turn out to being derived in the opposite way, namely from the orchestration to its called services. In this paper we propose an algorithm for probabilistic contract composition that the orchestration can run off-line, in support of negotiating contracts with its clients and called services.

New language feature: Finally, we also propose a new generic language feature that allows the orchestration to take decisions based on QoS con-

siderations. Examples include using the first response within a pool of called services, or preferring the most secure response among a pool of responses, or a composite thereof. This language feature is proposed on top of Orc, a clean and elegant language for orchestrations proposed by Cook and Misra [8] — this feature is indeed defined as a macro on top of core Orc.

The paper is organized as follows. Section 2 presents a detailed motivation for our theoretical developments. In particular, we discuss a simple but representative example of orchestration for which we provide a formal specification using our mild QoS-extension of Orc. Our formal model of *OrchNets* — a special class of colored occurrence nets — for the QoS study of orchestrations is presented in Section 3, where the algebra needed on QoS domains is developed and the competition policy is defined. Section 4 is devoted to monotonicity. We provide both probabilistic and non-probabilistic settings for it and show their relations; to this end, we make extensive use of the existing theory of *stochastic ordering* of random variables with values in a partially ordered domain [16]. Section 5 deals with probabilistic contracts and their composition. Experiments on contract composition are reported in Section 6 using a tool developed on top of the Orc language — Orc is a clean and elegant language for orchestrations proposed by Cook and Misra [8].

2 Informal study

In this section we first present a small although representative example of an orchestration and we use it to motivate QoS studies. In a second part we discuss on a toy example the underpinning mathematical issues.

2.1 The CarOnLine Example

2.1.1 Informal description

The CarOnLine orchestration is shown in Figure 1. In search for a second-hand car, a client calls the orchestration with an input car type — small car, family car, SUV, etc.

The orchestration calls two garages *GarageA* and *GarageB* in parallel, for getting quotes for the input car type. The calls to the garages are guarded by Timers. The action denoted by the small, bold square following the call to a garage and a timer selects the response with the "best QoS" w.r.t latency (*i.e.,* the shortest response time). Two other QoS parameters in the response of the garages are the car's price and the car's environmental friendliness that we assess using a *green level*.[1] Both the car price and the green level are used by the "*best*" action following the small bold squares, to select the best response amongst the two garages. CarOnLine then finds insurance and credit offers for the "*best*" offer. For credit offers, two services *AllCredit* and *AllCreditPlus* are called in parallel and the offer having the "*best*" (lower) interest rate is chosen. The insurance services called depends on the type of car which needs to be

[1] This refers to the current situation in France where a bonus/penalty system is attached to each car: when buying a clean car, you may get up to a 700 Euro bonus and you pay a penalty if the car is environment hostile. There are finitely many tax levels in this system.

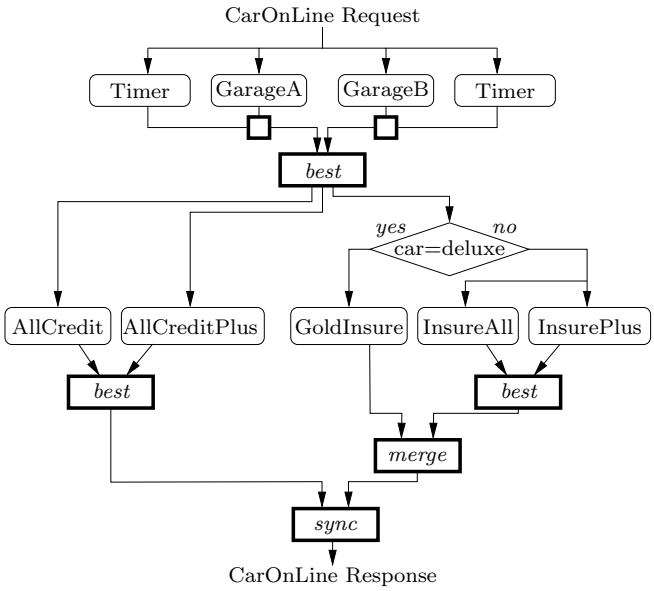

Figure 1: Schematic representation of the CarOnLine example. Calls to services are shown by rounded rectangles and processing actions internal to the orchestration are shown in bold boxes.

insured. If the car requested by the client is of some "deluxe" category, then only one service — *GoldInsure* — can offer insurance for such cars, and any offer made by it is taken. If the car is not a "deluxe" car, then two services, *InsureAll* and *InsurePlus* are called in parallel and the "*best*" insurance offer — the one that costs the least is selected.

In this example, the different attributes on which selection of the best response is performed, are seen as QoS parameters and the orchestration takes decisions based on the actual values of those QoS parameters. Each query to the orchestration comes as a token having data and QoS attributes. Each of these tokens are broadcast to the two garages and timers, which all produce a token at their output. These tokens are then combined when selecting the best offer. And so on. The traversal of any of the displayed sites by a token modifies both data and QoS attributes for the output tokens. Finally, one output token exits the orchestration for each given input token.

2.1.2 The Orc specification for CarOnLine

The program for CarOnLine is given in Table 1, using a mild extension of Orc for QoS management.

Orc [8] has three primitive operators. For Orc expressions f, g, "$f \mid g$" executes f and g in parallel. "$f >x> g$" evaluates f first and for *every* value returned by f, a *new* instance of g is launched with variable x assigned to this return value; in particular, "$f \gg g$" (which is a special case of the former where returned values are not assigned to any variable) causes *every* value returned by f to create a *new* instance of g. "f **where** $x :\in g$" executes f and g in parallel. When g returns its *first* value, x is assigned to this value and the computation

Assumptions QoS parameters :
δ : inter-query time, $D_\delta = \mathbb{R}_+$
Guarantees QoS parameters :
d : latency, $D_d = \mathbb{R}_+$
ℓ : green level, $D_\ell = \{0 \dots 4\}$
p : car price, $D_p = \mathbb{R}_+$
i : insurance costs, $D_i = \mathbb{R}_+$
c : credit rate, $D_c = \mathbb{R}_+$

$$CarOnLine(car) \triangleq CarPrice(car) >p> \textbf{let}(p, c, i)$$
$$\textbf{where } c :\in_d GetCredit(p)$$
$$i :\in_d GetInsur(p, car)$$

$$Best_Q(E_1, \dots, E_n) \triangleq \textbf{let}(a) \textbf{ where } a :\in_Q \{E_1 \mid E_2 \dots \mid E_n\}$$

$$CarPrice(car) \triangleq \{ Best_{(\ell,p)}(Best_d(GarageA[d, \ell, p](car), RTimer[d](T)),$$
$$Best_d(GarageB[d, \ell, p](car), RTimer[d](T)))$$
$$\} >p> \{ \textbf{if}(p \neq Fault)) \gg \textbf{let}(p) \}$$

$$GetCredit(p) \triangleq Best_c(AllCredit[d, c](p) \mid AllCreditPlus[d, c](p))$$

$$GetInsur(p, car) \triangleq \{ \textbf{if}(car = deluxe) \gg GoldInsure[d, i](p)\} \mid$$
$$\{ \textbf{if}(car \neq deluxe) \gg$$
$$Best_i(InsurePlus[d, i](p) \mid InsureAll[d, i](p))$$
$$\}$$

Table 1: CarOnLine in Orc, enhanced with QoS specification.

of g is terminated. All site calls in f having x as a parameter are blocked until x is defined (*i.e.,* until g returns its first value).

The support QoS management, we have proposed a new language feature for Orc, which we explain now. The services calls are enhanced with a list of the QoS parameters that the service acts on (for e.g., the $[d, l, p]$ in the call to *GarageA*).[2] The operator $:\in_Q$ is a new operator, where Q is the (static) parameter of this operator. Q is a QoS parameter whose domain is a partially ordered set $(D_Q, \leq)$; by convention, "best" will refer to a minimal element among a set. The expression "f **where** $x :\in_Q g$" does not take the first value returned by g as x. Instead it waits for a "best quality" response among all responses from g to that call, irrespective of the time taken to generate them — since the domain of Q is only partially ordered in general, a best response may not be unique. Observe that $:\in$ is a particular case of $:\in_Q$ by taking for Q the latency or response time of the call — in this case it is not needed to wait for all the responses from g to get the best one, since the first one received will, by definition, be the best.

2.2 The algebra of QoS computing, an informal discussion

In this section we discuss the algebra for computing QoS parameters using three operators for QoS increments, synchronization and competition. To this end, we use a mathematical model that captures the executions of a concurrent system, namely: Petri net unfoldings or occurrence nets. To represent QoS parameters, we enhance the tokens with colors, consisting of a pair

$$(v, q) \quad = \quad \text{(data, QoS value)} \tag{1}$$

[2]Some calls, like *CarPrice*, do not give a list of QoS parameters affected. This is because these are simply calls to Orc expressions, and not service calls.

Corresponding formal material will be developed in the next section.

A simple example with generic QoS parameter

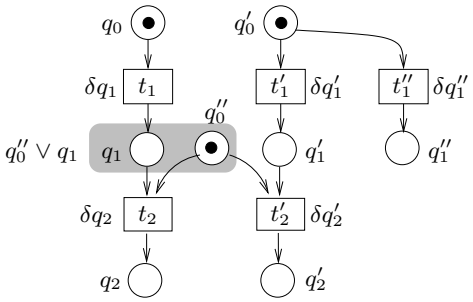

Figure 2: A simple example.

Figure 2 shows such a colored occurrence net, where only QoS values are mentioned — with no data. Each place comes labeled with a QoS value q which is the q-color of the token if it reaches that place.

QoS increments: When traversing a transition, each token has its QoS value incremented by some quantity that depends on the particular transition. For example, the left most token has initial QoS value q_0, which gets incremented as $q_1 = q_0 \oplus \delta q_1$ when traversing transition t_1, where $\oplus$ indicates adding increments.

Synchronizing tokens: A transition t is enabled when all places in its preset have tokens. For example, transition t_2 has input tokens with respective QoS values equal to $q_0 \oplus \delta q_1$ and q''_0. For the transition to fire, the two tokens must synchronize. The synchronization of these two tokens results in the "worst" QoS value, denoted by supremum $\vee$ associated to a given order $\leq$ used for synchronization, where smaller means better. When the two tokens sitting in the preset of transition t_2 get synchronized, the resulting synchronized pair has QoS value $q''_0 \vee q_1$. This is depicted on the figure with the shaded area.

Dealing with conflicts, competition policy: Let us first focus on the conflict following place q'_0. The QoS alters the usual semantics of the conflict by using a *competition policy* that is reminiscent of the classical race policy [**?**]. The competition between the two conflicting transitions in the post-set is solved by using order $\leq$ used for token synchronization. Thus we test whether $q'_o \oplus \delta q'_1 \leq q'_o \oplus \delta q''_1$ holds, or the converse. The smallest with respect to $\leq$ wins the competition — if equality holds, then nondeterministic choice occurs. However, comparing $q'_o \oplus \delta q'_1$ and $q'_o \oplus \delta q''_1$ generally requires knowing the two alternatives, which in turn can affect the QoS of the winner, as we shall see for specific QoS domains. This is taken into account by introducing a special operator $\lhd$.

More precisely, if two transitions t and t' are in competition and would yield tokens with respective QoS values q and q' in their post-sets, the cost of comparing them to set the competition alters the QoS value of the

winner in that — assuming the first wins — q is modified and becomes $q \lhd q'$. For the case of the figure, we get

$$\begin{array}{llll} \text{if} & (q'_o \oplus \delta q'_1) \leq (q'_o \oplus \delta q''_1) & \text{then } t'_1 \text{ fires and} & q'_1 = (q'_o \oplus \delta q'_1) \lhd (q'_o \oplus \delta q''_1) \\ \text{if} & (q'_o \oplus \delta q'_1) \geq (q'_o \oplus \delta q''_1) & \text{then } t''_1 \text{ fires and} & q''_1 = (q'_o \oplus \delta q''_1) \lhd (q'_o \oplus \delta q'_1) \end{array} \quad (2)$$

Now, another next conflict may occur between t_2 and t'_2. Now, if t''_1 actually wins the first competition, then t'_2 will never be enabled and this second potential conflict does not occur. In this case, t_2 fires and we get

$$q_2 = (q''_o \vee q_1) \oplus \delta q_2$$

where the first parenthesis involves the QoS after synchronizing the two input tokens of t_2 as shown by the shaded area. If, instead, t'_1 wins the first competition, then t'_2 will get enabled and this second potential conflict will occur. This conflict is then handled in a way similar to (2).

Some examples of QoS domains

We now review some examples of QoS domains. First, we introduce latency on responses and security. Then we introduce the less classical notion of "Quality of Data", for *e.g.,* "Quality of Response". Finally, we introduce composite QoS. For each of them, we instantiate the order $\leq$, and the two operators $\oplus$ (for QoS increments) and $\lhd$ (for handling competition policy).

1. *Latency*: QoS value of a token gives the accumulated latency, or "age" of the token since it was created when querying the orchestration, we denote it by d. The QoS domain is thus $\mathbb{R}_+$, equipped with $\oplus_d = +$, and $\leq_d =$ the usual order on $\mathbb{R}_+$ ($\vee$ is the usual *max* operator), which corresponds to using the race policy [7]. We now focus on operator $\lhd_d$. For the case of latency with race policy, comparing two dates via $d_1 \leq_d d_2$ does not impact the QoS of the winner: answer to this predicate is known as soon as one of the two events is seen, i.e., at time $\min(d, d')$.[3] Hence, for this case, we take $d_1 \lhd_d d_2 = d_1$, i.e., d_2 does not affect d_1. This is the basic example, which was studied in [5]. Since no generic QoS was considered in this reference, there was no need for considering $\lhd$.

2. *Security level*: QoS value s of a token belongs to ({high, low},$\leq_s$), with $high \leq_s low$. Each transition has a security level encoded in the same way, and we take $\oplus_s = \vee_s$, reflecting that a low security service processing a high security data yields a low security response. We now focus on operator $\lhd_s$. For this case also, comparing two dates via $s_1 \leq_s s_2$ does not impact the QoS of the winner, but reason for this differs from the previous case: QoS values are strictly "owned" by the tokens, and therefore do not interfere when comparing them. Hence, we take again $s_1 \lhd_s s_2 = s_1$, i.e., s_2 does not affect s_1.

3. *Green level* (cf. the CarOnLine example): $([0 \dots L], \leq, \vee)$, where 0 is the best value (lowest tax, or, equivalently, max bonus) and L the worst. Then we take, for the same reasons as for the previous case, $q \lhd q' = q$.

[3] This reflects the fact that the evaluation of $\min(d, d')$ can be done in a non-strict way.

4. *Quality of Response* is a generalization of the former. The value r of the token belongs to domain $(\mathbb{D}, \leq_r, \vee_r)$, where $\mathbb{D}$ is some finite set of labels. Examples for such a $\mathbb{D}$ are {valid, invalid}, {valid, exception$_1$,..., exception$_k$}, where valid $\leq_r$ exception$_i$ for $i = 1, \ldots, k$. Again, $r_1 \lhd_r r_2 = r_1$.

5. *Composite QoS, first example*: we may also consider a composite QoS parameter consisting of the pair (s, r) as above. When synchronizing tokens, the product order $\leq = \leq_s \times \leq_r$ is used, reflecting the fact that a low security level results from synchronizing a low security response with any other response, and a the same for quality of response. Alternatively, we may want to prioritize security; this is achieved by taking for $\leq$ the lexicographic order obtained from the pair $(\leq_s, \leq_r)$ by giving priority to s. For both cases, we take $\lhd = (\lhd_s, \lhd_r)$.

6. *Composite QoS, second example*: So far the special operator $\lhd$ did not play any role. We will need it, however, for the coming case, in which we consider a composite QoS parameter (s, d), where s and d are as above. We want to give priority to security s, and thus we now take $\leq$ to be the lexicographic order obtained from the pair $(\leq_s, \leq_d)$ by giving priority to s.

   Focus on operator $\lhd$. Consider the marking resulting after firing t_1 and t'_1 in figure 2, enabling t_2 and t'_2, which are in conflict. Let the QoS value of the token in postset of t_2, i.e. $q_2 = (low, d_2)$. (Recall that $q_2 = (q''_o \vee q_1) \oplus \delta q_2$.) Similarly, let $q'_2 = (low, d'_2)$ where $d'_2 >_d d_2$. ¿From the competition rule, transition t_2 wins the conflict and the outgoing token has QoS value $q_2 = (low, d_2)$. However, the decision to select t_2 can only be made when q'_2 is known, that is, at time d'_2. The reason for this is that, since at time d_2 a token with security level *low* is seen at place following t_2, it might be that a token with security level *high* later enters place following t'_2. The latter would win the conflict according to our competition policy — security level prevails. Observing that the right most token indeed has priority level *low* can only be seen at time d'_2. Thus it makes little sense assigning $q_2 = (low, d_2)$ to the outgoing token; it should rather be $q_2 = (low, d'_2)$. This is why a non-trivial operator $\lhd$ is needed, namely, writing $\leq$ for short instead of $\leq_d$:

$$(s, d) \lhd (s', d') \quad = \quad \textbf{if } d \leq d' \text{ and } s = low \textbf{ then } (s, d') \textbf{ else } (s, d) \quad (3)$$

Evaluation policy

It is worth summarizing, on the example of figure 2, the resulting semantics of the net, as enhanced with its QoS attributes. This is shown on figure 3. The shaded areas of the figure indicate the sub-nets which will never be activated. The formulas for q_1, q_2, etc, were given before.

Contracts: Assumptions versus Guarantees

So far we have only considered QoS offered by the called services and we have shown how to derive from them the QoS offered by the orchestration. In other words, we have only considered the obligations of the called services. But we

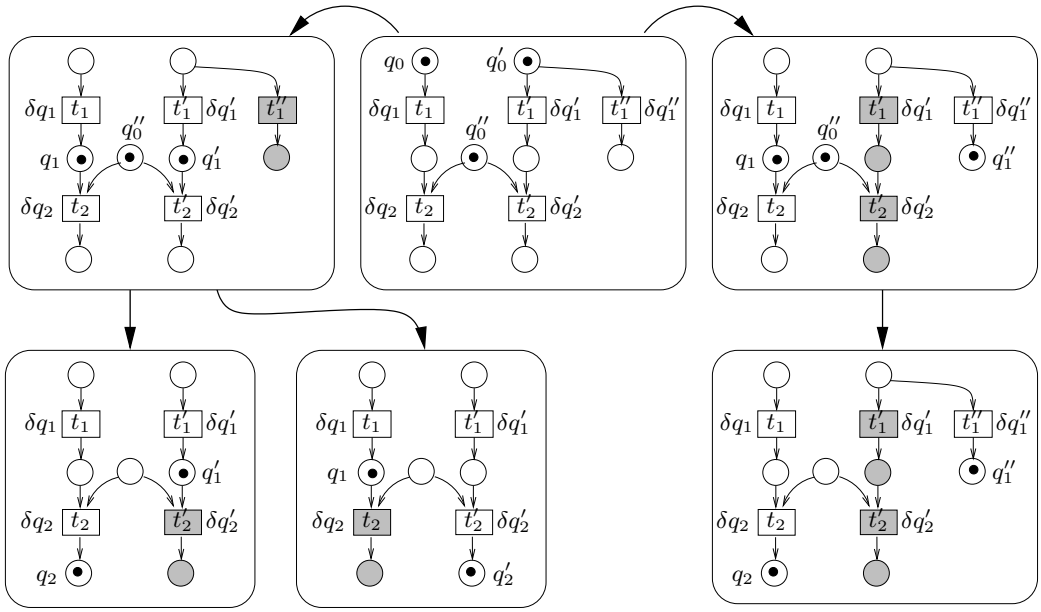

Figure 3: Evaluation steps of the example of figure 2.

have ignored the obligations of the agent calling the services — it can be the orchestration querying a service, or the customer querying the orchestration. For example, there is a limit in how many queries a same IP-address can call some free Web services. Another obligation is that the query must be well formed — although this is hardly considered as an issue of QoS. One can also imagine setting restrictions on the size of the parameters of a call to a service.[4]

Query throughput is directly measured by the called services, provided proper identifiers (for the calling client and considered query) are given. For our convenience, we will represent query throughput via the delay τ between two successive queries. Other QoS related characteristics of the queries are carried by the tokens entering the orchestration. To account for this, we augment the color (v, q) of (1) with a new attribute β characterizing the flow of queries. Thus we collect the pair $\alpha = (\tau, \beta)$ as an *assumption* to which each query is subject when submitted to a called service, and we denote by $\mathbb{A}$ the domain of assumptions.

Since we view QoS as benefiting from *guarantees* by the orchestration to its clients, and by the called services to the orchestration, characteristics α of the flow of queries will be in turn subject to *assumptions*. A *contract* will then be an implication

$$\textit{assumption} \Rightarrow \textit{guarantee}$$

expressing that some level of QoS is guaranteed by the orchestration to its client (or by a called service to the orchestration), provided that assumptions are met.

In contrast to QoS values, assumptions are modified by the orchestration, but indirectly as a consequence of control. For example, the time-and-data

[4]Observe that the same arise in QoS in the context of networks and IP. In this context, QoS involves parameters that are obligations for the network, e.g., jitter and latency, whereas not exceeding maximal throughput is an obligation for the user of the network.

dependent routing of tokens in the orchestration modifies the time elapsed since the previous query. However, in contrast to guarantees, no control flow decision is made while executing the orchestration based on assumptions.

3 The Orchestration Model: OrchNets

In this section we present *OrchNets* as our model for QoS enhanced orchestrations. OrchNets are a special form of *colored occurrence nets (CO-nets)* in which explicit provision is offered for QoS management. Occurrence nets are concurrent models of executions of Petri nets. They can support data values, QoS parameters, preemption, and recursion at no additional cost. Note that the executions of Workflow Nets [1, 18] are also CO-nets. We begin with the formal definition of QoS domains, for guarantees and assumptions.

3.1 QoS domains

Definition 1 (QoS domains for guarantees) *A* QoS domain for guarantees *is a tuple* $\mathbb{Q} = (\mathbb{D}, \leq, \oplus, \lhd)$ *where:*

$(\mathbb{D}, \leq)$ *is a partial order that is a complete upper lattice, meaning that every subset* $S \subseteq \mathbb{D}$ *has a unique least upper bound denoted by* $\bigvee S$*. By convention, we interpret synchronization order* $\leq$ *as "better". Hence operator* $\vee$ *amounts to taking the "worst" QoS and is used while synchronizing tokens.*

Operator $\oplus : \mathbb{D} \times \mathbb{D} \to \mathbb{D}$ *captures how a transition increments the QoS value; it satisfies the following conditions:*

1. *there exists some neutral element* 0 *with* $\forall q \in \mathbb{D} \Rightarrow q \oplus 0 = 0 \oplus q = q$*;*
2. $\oplus$ *is* monotonic*:*

$$q_1 \leq q_1' \quad \textit{and} \quad q_2 \leq q_2' \implies (q_1 \oplus q_2) \leq (q_1' \oplus q_2')$$

3. $\forall q, q' \in \mathbb{D}$, $\exists \delta q \in \mathbb{D}$ *such that* $q \leq q' \oplus \delta q$*.*

The competition function $\lhd : \mathbb{D} \times \mathbb{D}^* \to \mathbb{D}$*, where* $\mathbb{D}^* = \biguplus_{k=0}^{\infty} \mathbb{D}^k$ *and* $\mathbb{D}^0 = \emptyset$*, maps a pair consisting of 1/ the QoS resulting from the synchronization of the input tokens, and 2/ the tuple of the QoS of other tokens that must be considered when applying competition. We require the following regarding* $\lhd$*:*

1. $q \lhd \epsilon = q$ *where* ϵ *denotes the empty tuple, that is, if no competition occurs, then* q *is not altered;*
2. $\lhd$ *is monotonic:*

$$\begin{array}{c} q \leq q' \quad \textit{and} \quad q_1 \leq q_1', \ldots, q_n \leq q_n' \\ \Downarrow \\ (q \lhd (q_1, \ldots, q_n)) \leq (q' \lhd (q_1', \ldots, q_n')) \end{array}$$

Examples were given in section 2.2. The actual size of the second component of competition function $\lhd$ is dynamically determined while executing the net, this is why the domain of $\lhd$ is $\mathbb{D} \times \mathbb{D}^*$.

Definition 2 (QoS domains for Assumptions) *A QoS domain for assumption is a pair* $(\mathbb{A}, \leq^A)$*, where:* $\mathbb{A} = \mathbb{R}_+ \times \mathbb{B}$*,* $\mathbb{R}_+$ *is equipped with its usual order* $\leq$*,* $\mathbb{B}$ *is equipped with some partial order* $\leq_{\mathbb{B}}$*,* $\leq^A$ *is the product order, and* $(\mathbb{A}, \leq^A)$ *is a complete lower lattice, meaning that every subset* $S \subseteq \mathbb{D}$ *has a unique least lower bound denoted by* $\bigwedge S$*.*

For t a service, an assumption is a pair $\alpha = (\tau, \beta)$ where τ is the elapsed time since the last token was received, and β collects the other assumptions attached to the tokens. Observe that $\tau \leq \tau'$ must be interpreted as "τ is *worse* than τ'" from the point of view of t, because this amounts to increasing the load on t. We take the same interpretation for $\leq_{\mathbb{B}}$, and also for partial order $\leq^A$ on $\mathbb{A}$; thus "better" translates as "$\geq^A$" for assumptions, which is the converse of guarantees.

If some QoS parameter q of the orchestration is irrelevant to a service it involves, we take the convention that this service acts on tokens with a 0 increment on the value of q. With this convention we can safely assume that the orchestration, all its called services, and all its tokens use the same QoS domain. This assumption will be in force in the sequel.

Before providing the formal definition of OrchNets, we need some background on occurrence nets.

3.2 Background on Petri nets and Occurrence nets

We assume that the reader is familiar with the basics of Petri nets [9]. A *Petri net* is a tuple $N = (\mathcal{P}, \mathcal{T}, \mathcal{F}, M_0)$, where: $\mathcal{P}$ is a set of *places*, $\mathcal{T}$ is a set of *transitions* such that $\mathcal{P} \cap \mathcal{T} = \emptyset$, $\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ is the *flow relation*, $M_0 : \mathcal{P} \to \mathbf{N}$ is the *initial marking*. For $x \in \mathcal{P} \cup \mathcal{T}$, we call ${}^\bullet x = \{y \mid (y, x) \in \mathcal{F}\}$ the *preset* of x, and $x^\bullet = \{y \mid (x, y) \in \mathcal{F}\}$ the *postset* of x. For a net $N = (\mathcal{P}, \mathcal{T}, \mathcal{F}, M_0)$ the *causality relation* $\leq$ is the transitive and reflexive closure of $\mathcal{F}$. For a node $x \in \mathcal{P} \cup \mathcal{T}$, the set of *causes* of x is $\lceil x \rceil = \{y \in \mathcal{P} \cup \mathcal{T} \mid y \leq x\}$. Two nodes x and y are in *conflict*, denoted by $x \# y$, if there exist distinct transitions $t, t' \in T$, such that $t \leq x, t' \leq y$ and ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$. Nodes x and y are said to be *concurrent* if neither $(x \leq y)$ nor $(y \leq x)$ nor $(x \# y)$. A *configuration* of N is a subnet κ of nodes of N such that: 1/ if $x < x'$ and $x' \in \kappa$ then $x \in \kappa$; and, 2/ for all nodes $x, x' \in \kappa, \neg(x \# x')$. For convenience, we require that the maximal nodes in a configuration are places.

Occurrence nets: A Petri net is *safe* if all its reachable markings M satisfy $M(\mathcal{P}) \subseteq \{0, 1\}$. A safe net $N = (\mathcal{P}, \mathcal{T}, \mathcal{F}, M_0)$ is an *occurrence net (O-net)* iff

1. $\neg(x \# x)$ for every $x \in \mathcal{P} \cup \mathcal{T}$;
2. $\leq$ is a partial order and $\lceil t \rceil$ is finite for any $t \in \mathcal{T}$;
3. for each place $p \in \mathcal{P}$, $|{}^\bullet p| \leq 1$;
4. $M_0 = \{p \in \mathcal{P} | {}^\bullet p = \emptyset\}$ holds.

Occurrence nets are a good model for representing the possible executions of a concurrent system.

Branching cells: The discussion of the example in section 2.2 revealed the need to consider, dynamically while execution progresses, the set of transitions that are both enabled and in conflict with a considered transition. This was studied by S. Abbes and A. Benveniste with the notion of branching cell [2], which we recall now. Let N be an occurrence net. Two transitions $t, t' \in \mathcal{T}$ are in *minimal conflict*, written $t \#_m t'$, if and only if $(\lceil t \rceil \times \lceil t' \rceil) \cap \# = \{(t, t')\}$, where $\lceil t \rceil = \{s \in \mathcal{T} \mid s \leq t\}$ is the set of transitions causing t. A prefix M of N is a causally closed subnet of N whose maximal nodes are places; formally, M is closed under operations $t \to \lceil t \rceil$ and $t \to t^\bullet$. Prefix M is called a *stopping prefix* if it is closed under minimal conflict: $t \in M$ and $t' \#_m t$ imply $t' \in M$. *Branching cells* of occurrence net N are inductively defined as follows: 1/ every minimal (for prefix relation) stopping prefix of N is a branching cell, and, 2/ let B be any such branching cell and κ any maximal configuration of it, then any branching cell of N^κ is a branching cell of N, where N^κ, *the future of* κ, is defined by

$$N^\kappa = \{x \in N \setminus \kappa \mid \forall x' \in \kappa, \neg(x \# x')\} \cup \mathit{maxPlaces}(\kappa) \tag{4}$$

where $\mathit{maxPlaces}(\kappa)$ is the set of maximal nodes of κ (which are all places). [5] A result regarding branching cells that we will need is that *the minimal* (for causality order) *branching cells of an occurrence net are pairwise concurrent.*

3.3 OrchNets: formal definition and semantics

In this section we assume QoS domains $(\mathbb{D}, \leq, \oplus, \lhd)$, for guarantees, and $(\mathbb{A}, \leq^A)$, for assumptions.

Definition 3 (OrchNet) *An* OrchNet *is a tuple* $\mathcal{N} = (N, V, A, Q, Q_{\text{init}})$ *consisting of*

A finite occurrence net N *with token attributes*

$$c = (v, \beta, q) = (\text{data, assumption, QoS value})$$

A family $V = (\nu_t)_{t \in \mathcal{T}}$ *of* value functions, *mapping the data values of the transition's input tokens to the data value of the transition's output token.*

A family $A = (\alpha_t)_{t \in \mathcal{T}}$ *of* assumptions, *where* $\alpha_t = (\tau_t, \beta_t)$ *for each* $t \in \mathcal{T}$, τ_t *is the time elapsed since the previous token traversed transition* t, *and* β_t *is the value set for the assumptions by transition* t *when the considered token traverses it.*

A family $Q = (\xi_t)_{t \in \mathcal{T}}$ *of* QoS functions, *mapping the data values of the transition's input tokens to a QoS increment.*

A family $Q_{\text{init}} = (\xi_p)_{p \in \min(\mathcal{P})}$ *of* initial QoS functions *for the minimal places of* N.

Values, assumptions, and QoS functions can be nondeterministic. We introduce a global, invisible, daemon variable ω *that resolves this nondeterminism and we denote by* Ω *its domain. That is, for* $\omega \in \Omega$, $\nu_t(\omega), \alpha_t(\omega), \xi_t(\omega)$, *and* $\xi_p(\omega)$ *are all deterministic functions of their respective inputs.*

[5] In the example of figure 2, transitions t_1' and t_1'', along with their pre and post sets form one of the branching cells of the net.

Competition Policy

We will now explain how the presence of QoS values attached to tokens affects the semantics of OrchNets. Assumptions play no role in the competition policy, see our discussion of section 2.2. Thus, in the following analysis, we can safely ignore α. So, when talking about a "QoS value" in this subsection, we mean a QoS value for guarantees. Accordingly, we will consider that any place p of occurrence net N has a pair $(v_p, q_p) =$ (data, QoS value) assigned to it, which is the color held by a token reaching that place.

Procedure 1 (competition policy) *Let $\omega \in \Omega$ be any value for the daemon. The continuation of any finite configuration $\kappa(\omega)$ is constructed by performing the following steps, where we omit the explicit dependency of $\kappa(\omega)$, $\nu_t(\omega)$, and $\xi_t(\omega)$, with respect to ω, for the sake of clarity:*

1. *Choose nondeterministically a minimal branching cell B in the future of κ.*
2. *For t any minimal transition of B, compute:*

$$q_t \quad = \quad \left(\bigvee_{p' \in {}^\bullet t} q_{p'}\right) \oplus \xi_t(v_{p'} \mid p' \in {}^\bullet t) \tag{5}$$

3. *Competition step: select nondeterministically a minimal transition t_* of B such that no other minimal transition t of B exists such that $q_t < q_{t_*}$.*
4. *Augment κ to $\kappa' = \kappa \cup \{t_*\} \cup t_*^\bullet$, and assign, to every $p \in t_*^\bullet$, the pair (v, q), where*

$$\begin{array}{rcl} v & = & \nu_t(v_{p'} \mid p' \in {}^\bullet t) \\ q & = & q_{t_*} \lhd (q_t \mid t \in B, t \text{ minimal}, t \neq t_*) \end{array} \tag{6}$$

Observe that the augmented configuration κ' as well as the pair (v, q) are dependent on ω.

Step 4 of competition policy simplifies for the examples 1–5 of section 2.2, since $q \lhd (q_1, \ldots, q_n) = q$ in these cases. On the other hand, a non-trivial operator $\lhd$ was needed to address example 6, see formula (3).

Since occurrence net N is finite, the competition policy terminates in finitely many steps when $N^{\kappa(\omega)} = \emptyset$. The total execution thus proceeds by a finite chain of nested configurations: $\emptyset = \kappa_0(\omega) \subset \kappa_1(\omega) \cdots \subset \kappa_n(\omega)$. Hence, $\kappa_n(\omega)$ is a maximal configuration of $\mathcal{N}$ that can actually occur according to the competition policy, for a given $\omega \in \Omega$; we generically denote it by

$$\kappa(\mathcal{N}, \omega). \tag{7}$$

For the example 1 of latency, our competition policy boils down to the classical *race policy* [7]. Our competition policy bears some similarity with the "preselection policies" introduced in [7], except that the continuation is selected based on QoS values in our case, not on random selection. We will also need to compute the QoS for *any* configuration of N, even if it is not a winner of the competition policy. We do this using procedure 1, but without the competition step:

Procedure 2 (QoS of an arbitrary configuration) *Let $\kappa_{\max}$ be any maximal configuration of N and $\kappa \subseteq \kappa_{\max}$ a prefix of it. With reference to procedure 1, perform the following: step 1 with B any minimal branching cell in $\kappa_{\max} \setminus \kappa$, step 2 with no change, and then step 4 for any t as in step 2. Performing this repeatedly yields the pair (v_p, q_p) for each place p of $\kappa_{\max}$.*

We are now ready to define what the QoS value of an OrchNet is, thus formalizing what we mean by "the QoS of an orchestration".

Definition 4 (end-to-end QoS) *For κ any configuration of occurrence net N, and ω any value for the daemon, the* end-to-end QoS *of κ is defined as*

$$E_\omega(\kappa, \mathcal{N}) = \bigvee_{p \in maxPlaces(\kappa)} q_p(\omega) \tag{8}$$

The end-to-end QoS *and* loose end-to-end QoS *of OrchNet $\mathcal{N}$ are given by*

$$\text{end-to-end QoS} : E_\omega(\mathcal{N}) = E_\omega(\kappa(\mathcal{N}, \omega), \mathcal{N}) \tag{9}$$

$$\text{loose end-to-end QoS} : F_\omega(\mathcal{N}) = \max\{E_\omega(\kappa, \mathcal{N}) \mid \kappa \in \mathcal{V}(N)\} \tag{10}$$

where function max *picks one of the maximal values in a partially ordered set, $\kappa(\mathcal{N}, \omega)$ is defined in (7), and $\mathcal{V}(N)$ is the set of all maximal configurations of net N.*

Observe that $E_\omega(\mathcal{N}) \leq F_\omega(\mathcal{N})$ holds and $E_\omega(\mathcal{N})$ is indeed observed when the orchestration is executed. The reason for considering in addition $F_\omega(\mathcal{N})$ will be made clear in the next section on monotonicity.

4 Study of Monotonicity

In this section we focus on monotonicity of (loose) end-to-end QoS for an orchestration. We wish to formalize that a given orchestration is QoS-monotonic: "if any called service performs better, then so will the orchestration do".

4.1 Defining and characterizing monotonicity

To formalize monotonicity we must be able to specify how QoS increments can vary. As an example, we may want to constrain some pair of transitions to have identical QoS increments. This can be stated by specifying a legal set of families of QoS functions. For example, this legal set may accept any family $Q = (\xi_t)_{t \in \mathcal{T}}$ such that two given transitions t and t' possess equal QoS increments: $\forall \omega \Rightarrow \xi_t(\omega) = \xi_{t'}(\omega)$. The same technique can be used for initial QoS values. Thus, the flexibility in setting QoS increments and initial QoS values can be formalized under the notion of pre-OrchNet we introduce next.[6]

Definition 5 (pre-OrchNet) *Call* pre-OrchNet *a tuple $\mathbb{N} = (N, \Phi, \mathbb{Q}, \mathbb{Q}_{\text{init}})$, where N and Φ are as before, and $\mathbb{Q}$ and $\mathbb{Q}_{\text{init}}$ are sets of families Q of QoS functions and of families Q_{init} of initial QoS functions. Write $\mathcal{N} \in \mathbb{N}$ if $\mathcal{N} = (N, \Phi, Q, Q_{\text{init}})$ for some $Q \in \mathbb{Q}$ and $Q_{\text{init}} \in \mathbb{Q}_{\text{init}}$.*

[6] In a shorter version of this paper [15], monotonicity is studied in the simpler case where no such constraint can be stated. The theory simplifies a lot and there is no need to introduce pre-OrchNets. Conditions for monotonicity remain essentially the same, however.

For two families Q and Q' of QoS functions, write

$$Q \geq Q'$$

to mean that $\forall \omega \in \Omega, \forall t \in \mathcal{T} \Rightarrow \xi_t(\omega) \geq \xi'_t(\omega)$, and similarly for $Q_{\text{init}} \geq Q'_{\text{init}}$. For $\mathcal{N}, \mathcal{N}' \in \mathbb{N}$, write

$$\mathcal{N} \geq \mathcal{N}' \,,\; E(\mathcal{N}) \geq E(\mathcal{N}') \,, \quad \text{and} \quad F(\mathcal{N}) \geq F(\mathcal{N}')$$

to mean that $Q \geq Q'$ and $Q_{\text{init}} \geq Q'_{\text{init}}$ both hold, and $E_\omega(\mathcal{N}) \geq E_\omega(\mathcal{N}')$ and $F_\omega(\mathcal{N}) \geq F_\omega(\mathcal{N}')$ holds for every ω, respectively.

Definition 6 (monotonicity) *pre-OrchNet* $\mathbb{N} = (N, \Phi, \mathbb{Q}, \mathbb{Q}_{\text{init}})$ *is called* monotonic (*resp.* loosely monotonic) *if, for any two* $\mathcal{N}, \mathcal{N}' \in \mathbb{N}$*, such that* $\mathcal{N} \geq \mathcal{N}'$*, then* $E(\mathcal{N}) \geq E(\mathcal{N}')$ (*resp.* $F(\mathcal{N}) \geq F(\mathcal{N}')$) *holds.*

The following immediate result justifies considering loose end-to-end QoS $F(\mathcal{N})$ in addition to end-to-end QoS $E(\mathcal{N})$:

Theorem 1 (loose monotonicity) *Any pre-OrchNet is loosely monotonic.*

Consequently, it is always sound to base contract composition — see later — and contract monitoring [**?**] on loose end-to-end QoS. This, however, has a price, since loose end-to-end QoS is pessimistic compared to (actual) end-to-end QoS.

The next theorem gives conditions ensuring monotonicity, i.e., based on tight end-to-end QoS $E(\mathcal{N})$:

Theorem 2 (monotonicity: global necessary and sufficient condition) *1.*

The following implies the monotonicity of pre-OrchNet $\mathbb{N} = (N, \Phi, \mathbb{Q}, \mathbb{Q}_{\text{init}})$*:*

$$\forall \mathcal{N} \in \mathbb{N}, \forall \omega \in \Omega, \forall \kappa \in \mathcal{V}(N) \quad \Longrightarrow \quad E_\omega(\kappa, \mathcal{N}) \geq E_\omega(\kappa(\mathcal{N}, \omega), \mathcal{N}) \tag{11}$$

where $\mathcal{V}(N)$ *denotes the set of all maximal configurations of net* N *and* $\kappa(\mathcal{N}, \omega)$ *is the maximal configuration of* $\mathcal{N}$ *that actually occurs under the daemon value* ω*.*

2. Conversely, assume that:

(a) Condition (11) is violated, and

(b) for any two OrchNets $\mathcal{N}$ *and* $\mathcal{N}'$ *s.t.* $\mathcal{N} \in \mathbb{N}$*, then* $\mathcal{N}' \geq \mathcal{N} \Rightarrow \mathcal{N}' \in \mathbb{N}$*.*

Then $\mathbb{N} = (N, \Phi, \mathbb{Q}, \mathbb{Q}_{\text{init}})$ *is not monotonic.*

Statement 2 expresses that Condition (11) is also necessary provided that it is legal to increase at will QoS increments or initial QoS functions. Observe that violating Condition (11) is by itself not a cause of lack of monotonicity; as a counterexample, consider a case where $\mathbb{Q}$ is a singleton for which (11) is violated — it is nevertheless monotonic.

4.2 A structural condition for the monotonicity of workflow nets

Workflow nets [18] were proposed as a simple model for workflows. These are Petri nets, with a special minimal place i and a special maximal place o. We consider the class of workflow nets that are 1-safe and which have no loops. Further, we require them to be *sound* [18]. A Workflow net W is *sound* iff:

1. For every marking M reachable from the initial place i, there is a firing sequence leading to the final place o.
2. If a marking M marks the final place o, then no other place can in W can be marked in M
3. There are no *dead* transitions in W. Starting from the initial place, it is always possible to fire any transition of W.

Workflow nets will be generically denoted by W. We can equip workflow nets with the same attributes as occurrence nets, thus defining *pre-WFnets* $\mathbb{W} = (W, \Phi, \mathbb{Q}, \mathbb{Q}_{\text{init}})$. Referring to the end of Section 3.2, unfolding W yields an occurrence net that we denote by N_W with associated morphism $\varphi_W : N_W \mapsto W$. Here the morphism φ_W maps the two c transitions (and the place in its preset and postset) in the net on the right to the single c transition (and its preset and postset) in the net on the left. Observe that W and N_W possess identical sets of minimal places. Morphism φ_W induces a pre-OrchNet

$$\mathbb{N}_W = (N_W, \Phi_W, \mathbb{Q}_W, \mathbb{Q}_{\text{init}})$$

by attaching to each transition t of N_W the value and QoS functions attached to $\varphi_W(t)$ in $\mathbb{W}$.

We shall use the results of the previous section in order to characterize those pre-WFnets whose unfoldings give monotonic pre-OrchNets. Our characterization will be essentially structural in that it does not involve any constraint on QoS functions. Under this restricted discipline, the simple structural conditions we shall formulate will also be almost necessary. For this, we recall a notion of *cluster* [9] on nets. For a net N, a *cluster* is a minimal set $\mathbf{c}$ of places and transitions of N such that $\forall t \in \mathbf{c},\ {}^\bullet t \subseteq \mathbf{c}$ and $\forall p \in \mathbf{c},\ p^\bullet \subseteq \mathbf{c}$.

Theorem 3 (Sufficient Condition) *Let W be a WFnet and N_W be its unfolding. A sufficient condition for the pre-OrchNet $\mathbb{N}_W = (N_W, \Phi_W, \mathbb{Q}_W, \mathbb{Q}_{\text{init}})$ to be monotonic is that every cluster $\mathbf{c}$ satisfies the following condition:*

$$\forall t_1, t_2 \in \mathbf{c},\ t_1 \neq t_2 \implies t_1^\bullet = t_2^\bullet \tag{12}$$

Recall that the sufficient condition for monotonicity stated in Theorem 2 is "almost necessary" in that, if enough flexibility exist in setting QoS increments and initial QoS values, then it is actually necessary. The same holds for the sufficient condition stated in Theorem 3 if the workflow net is assumed to be live.

Theorem 4 (Necessary Condition) *Assume that the workflow net W is sound. Assume that $\mathcal{W} \in \mathbb{W}$ and $\mathcal{W}' \geq \mathcal{W}$ implies $\mathcal{W}' \in \mathbb{W}$, meaning that there is*

enough flexibility in setting QoS increments and initial QoS values. In addition, assume that partial order $(\mathbb{D}, \leq)$ *is such that*

$$\forall q \in \mathbb{D}, \exists q' \in \mathbb{D} \implies q' > q \tag{13}$$

Then the sufficient condition of Theorem 3 is also necessary for monotonicity.

4.3 Probabilistic monotonicity

So far we have considered the case where QoS increments of transitions are nondeterministic. In [12, 13], however, we have advocated the use of probability distributions when modeling the response time of a Web service. Consequently, monotonicity for probabilistic QoS was studied in [5] for the restricted case of latency only. Here we further extend the range of this study to the general framework for QoS studies developed in this paper.

As a prerequisite, we need some background on probabilistic ordering — more usually called *stochastic ordering* [16]. We first recall the classical case when $(\mathbb{D}, \leq)$ is a total order: If X is a random variable with values in $\mathbb{D}$, its *distribution function* is defined by $F(x) = \mathbf{P}(X \leq x)$, where $x \in \mathbb{D}$. Consider the following ordering on probability distributions: random variables X and X' satisfy $X \geq_s X'$ if $F(x) \leq F'(x)$ holds $\forall x \in \mathbb{D}$, where F and F' are the distribution functions of X and X', respectively. The intuition is that "there are bigger chances that X' is smaller than x as compared to X". This order is classical in probability theory, where it is referred to as *stochastic dominance* or *stochastic ordering* among random variables [16].

To extend stochastic dominance to the case where $\leq$ is only a partial order, not a total order, we proceed as follows. Consider *ideals* of $\mathbb{D}$, i.e., subsets I of $\mathbb{D}$ that are downward closed: $x \in I$ and $y \leq x \implies y \in I$. Examples of ideals are: for $\mathbb{R}_+$, the intervals, $[0, x]$ for all x; for $\mathbb{R}_+ \times \mathbb{R}_+$ equipped with the product order, arbitrary unions of rectangles $[0, x] \times [0, y]$. Now, if X has values in $\mathbb{D}$, we define its *distribution function* by

$$F(I) = \mathbf{P}(X \in I), \tag{14}$$

for I ranging over the set of all ideals of $\mathbb{D}$. For X and X' two random variables with values in $\mathbb{D}$, with respective distribution functions F and F', define

$$X \geq_s X' \text{ iff } \quad \text{for any ideal } I \text{ of } \mathbb{D} \Rightarrow F(I) \leq F'(I) \tag{15}$$

4.3.1 Probabilistic setting, first attempt

In Definitions 3 and 5, QoS and initial QoS functions were considered nondeterministic. The first idea is to let them become random instead. This leads to the following straightforward modification of definitions 3 and 5:

Definition 7 (probabilistic OrchNet and pre-OrchNet, 1) *Call* probabilistic OrchNet *a tuple* $\mathcal{N} = (N, \Phi, Q, Q_{\text{init}})$ *where* $\Phi = (\phi_t)_{t \in \mathcal{T}}$, $Q = (\xi_t)_{t \in \mathcal{T}}$, *and* $Q_{\text{init}} = (\xi_p)_{p \in \min(\mathcal{P})}$, *are independent families of* random *value functions, QoS functions, and initial QoS functions, respectively.*

Call probabilistic pre-OrchNet *a tuple* $\mathbb{N} = (N, \Phi, \mathbb{Q}, \mathbb{Q}_{\text{init}})$, *where* N *and* Φ *are as before, and* $\mathbb{Q}$ *and* $\mathbb{Q}_{\text{init}}$ *are sets of families* T *of* random *QoS functions and of families* Q_{init} *of* random *initial QoS functions. Write* $\mathcal{N} \in \mathbb{N}$ *if* $\mathcal{N} = (N, \Phi, Q, Q_{\text{init}})$ *for some* $T \in \mathbb{Q}$ *and* $Q_{\text{init}} \in \mathbb{Q}_{\text{init}}$.

We now equip random QoS increments and initial QoS values with a probabilistic ordering associated to order $\leq$.

Using order (15), for two families $Q = (\xi_t)_{t\in\mathcal{T}}$ and $Q' = (\xi'_t)_{t\in\mathcal{T}}$ of QoS functions, write

$$Q \geq_s Q'$$

to mean that $\forall t \in \mathcal{T} \implies \xi_t \geq_s \xi'_t$, and similarly for $Q_{\text{init}} \geq_s Q'_{\text{init}}$. For $\mathcal{N}, \mathcal{N}' \in \mathbb{N}$, write

$$\mathcal{N} \geq_s \mathcal{N}'$$

if $T \geq_s T'$ and $Q_{\text{init}} \geq_s Q'_{\text{init}}$ both hold. Finally, the latency $E_\omega(\mathcal{N})$ of OrchNet $\mathcal{N}$ is itself seen as a random variable that we denote by $E(\mathcal{N})$, by removing symbol ω. This allows us to define, for any two $\mathcal{N}, \mathcal{N}' \in \mathbb{N}$,

$$E(\mathcal{N}) \geq_s E(\mathcal{N}')$$

by requiring that random variables $E(\mathcal{N})$ and $E(\mathcal{N}')$ are stochastically ordered.

Definition 8 (probabilistic monotonicity, 1) *Probabilistic pre-OrchNet* $\mathbb{N}$ *is called* probabilistically monotonic *if, for two* $\mathcal{N}, \mathcal{N}' \in \mathbb{N}$ *such that* $\mathcal{N} \geq_s \mathcal{N}'$*, we have* $E(\mathcal{N}) \geq_s E(\mathcal{N}')$*.*

It is a classical result on stochastic ordering that, if $(X_1, \ldots, X_n)$ and $(Y_1, \ldots, Y_n)$ are independent families of real-valued random variables such that $X_i \geq_s Y_i$ for every $1 \leq i \leq n$, then, for any increasing function $f : \mathbb{R}^n \to \mathbb{R}$, then $f(X_1, \ldots, X_n) \geq_s f(Y_1, \ldots, Y_n)$. Applying this yields that nondeterministic monotonicity in the sense of definition 6 implies probabilistic monotonicity in the sense of to definition 8. Nothing can be said, however, regarding the converse.

In order to derive results in the opposite direction, we shall establish a tighter link between this probabilistic framework and the nondeterministic framework of sections 3 and 4.

4.3.2 Probabilistic setting: second attempt

Let us restart from the nondeterministic setting of sections 3 and 4. Focus on definition 3 of OrchNets. Equipping the set Ω of all possible values for the daemon with a probability $\mathbf{P}$ yields an alternative way to make the QoS functions and initial QoS random. This suggests the following alternative setting for probabilistic monotonicity.

Definition 9 (probabilistic OrchNet and pre-OrchNet, 2) *Call* probabilistic OrchNet *a pair* $(\mathcal{N}, \mathbf{P})$*, where* $\mathcal{N}$ *is an OrchNet according to definition 3 and* $\mathbf{P}$ *is a probability over the domain* Ω *of all values for the daemon.*

Call probabilistic pre-OrchNet *a pair* $(\mathbb{N}, \mathbf{P})$*, where* $\mathbb{N}$ *is a pre-OrchNet according to definition 5 and* $\mathbf{P}$ *is a probability over the domain* Ω *of all values for the daemon.*

How can we relate the two definitions 7 and 9? Consider the following assumption, which will be in force in the sequel:

Assumption 1 *For any $\mathcal{N} \in \mathbb{N}$, ξ_t and ξ_p form an independent family of random variables, for t ranging over the set of all transitions and p ranging over the set of all minimal places of the underlying net.*

Let us now start from definition 7. For t a generic transition, let $(\Omega_t, \mathbf{P}_t)$ be the set of possible experiments together with associated probability, for random latency ξ_t; and similarly for $(\Omega_p, \mathbf{P}_p)$ and ξ_p. Thanks to assumption 1, setting

$$\Omega = (\prod_t \Omega_t) \times (\prod_p \Omega_p) \quad \text{and} \quad \mathbf{P} = (\prod_t \mathbf{P}_t) \times (\prod_p \mathbf{P}_p), \tag{16}$$

yields the entities of definition 9. Can we use this correspondence to further relate probabilistic monotonicity to the notion of monotonicity of sections 3 and 4? In the nondeterministic framework of section 4, definition 5, we said that

$$\xi \geq \xi' \text{ if } \xi(\omega) \geq \xi'(\omega) \text{ holds } \forall \omega \in \Omega, \tag{17}$$

Clearly, if two random latencies ξ and ξ' satisfy condition (17), then they also satisfy condition (15). That is, *ordering* (17) *is stronger than stochastic ordering* (15). Unfortunately, the converse is *not* true in general. For example, condition (15) may hold while ξ and ξ' are two independent random variables, which prevents (17) from being satisfied. Nonetheless, the following result holds, which will allow to proceed:

Theorem 5 *Assume condition* (15) *holds for the two distribution functions F and F'. Then, there exists a probability space Ω, a probability $\mathbf{P}$ over Ω, and two real valued random variables $\hat{\xi}$ and $\hat{\xi}'$ over Ω, such that:*

1. *$\hat{\xi}$ and $\hat{\xi}'$ possess F and F' as respective distribution functions, and*
2. *condition* (17) *is satisfied by the pair $(\hat{\xi}, \hat{\xi}')$ with probability* 1.

The proof of this result is immediate if $(\mathbb{D}, \leq)$ is a total order. It is, however, highly nontrivial if $\leq$ is only a partial order. This theorem is indeed part of theorem 1 of [16].[7]

Theorem 5 allows to reduce the stochastic comparison of random variables to their ordinary comparison as functions defined over the same set of experiments endowed with a same probability. This applies in particular to each random QoS function and each random initial QoS function, when considered in isolation. Thus, when performing construction (16) for two OrchNets $\mathcal{N}$ and $\mathcal{N}'$, we can take the *same* pair $(\Omega_t, \mathbf{P}_t)$ to represent both ξ_t and ξ'_t, and similarly for ξ_p and ξ'_p. Applying (16) implies that both $\mathcal{N}$ and $\mathcal{N}'$ are represented using the *same* pair $(\Omega, \mathbf{P})$. This leads naturally to definition 9.

In addition, applying theorem 5 to each transition t and each minimal place p yields that stochastic ordering $\mathcal{N} \geq_s \mathcal{N}'$ reduces to ordinary ordering $\mathcal{N} \geq \mathcal{N}'$. Observe that this trick does not apply to the overall latencies $E(\mathcal{N})$ and $E(\mathcal{N}')$ of the two OrchNets; the reason for this is that the space of experiments for these two random variables is already fixed (it is Ω) and cannot further be played with as theorem 5 requires. Thus we can reformulate probabilistic monotonicity as follows — compare with definition 8:

[7] Thanks are due to Bernard Delyon who pointed this reference to us.

Definition 10 (probabilistic monotonicity, 2) *Probabilistic pre-OrchNet* $(\mathbb{N}, \mathbf{P})$ *is called* probabilistically monotonic *if, for any two* $\mathcal{N}, \mathcal{N}' \in \mathbb{N}$*, such that* $\mathcal{N} \geq \mathcal{N}'$*, we have* $E(\mathcal{N}) \geq_s E(\mathcal{N}')$*.*

Note the careful use of $\geq$ and $\geq_s$. The following two results establish a relation between probabilistic monotonicity and monotonicity:

Theorem 6 *If pre-OrchNet* $\mathbb{N}$ *is monotonic, then, probabilistic pre-OrchNet* $(\mathbb{N}, \mathbf{P})$ *is probabilistically monotonic for any probability* $\mathbf{P}$ *over the set* Ω*.*

This result was already obtained in the first probabilistic setting; it is here a direct consequence of the fact that $\xi \geq \xi'$ implies $\xi \geq_s \xi'$ if ξ and ξ' are two random variables defined over the same probability space. The following converse result completes the landscape and is much less straightforward. It assumes that it is legal to increase at will latencies or initial dates, see theorem 2:

Theorem 7 *Assume that condition 2b of theorem 2 is satisfied. Then, if probabilistic pre-OrchNet* $(\mathbb{N}, \mathbf{P})$ *is probabilistically monotonic, it is also monotonic with* $\mathbf{P}$*-probability* 1*.*

5 Probabilistic contracts and their composition

In this section we study probabilistic contracts and their composition. We thus consider probabilistic OrchNets and pre-OrchNets according to definition 7. In this context, for any given service, a probabilistic contract will be a pair consisting of a probabilistic obligation regarding QoS, and a probabilistic assumption that the client of this service must comply with. Formally,

Definition 11 (probabilistic contracts) *A* contract *is a pair* (F_A, F_Q) *of cumulative distribution functions, representing the assumed distribution for the random assumption* α *and the guaranteed distribution for the random QoS function* ξ*.*

To define what it means to satisfy a contract, we shall use stochastic ordering introduced in (15). By abuse of language, we will sometimes write $F \geq_s F'$ or even $X \geq_s F'$ instead of $X \geq_s X'$. Using (15), we can now define what it means for a random pair $(\alpha, \xi) =$ (assumption, QoS value), to satisfy a contract:

Definition 12 (satisfaction) *Pair* (α, ξ)*, consisting of an actual random parameter for the assumptions, and an actual random value for the QoS,* satisfies contract (F_A, F_Q) *if* $\alpha \geq_s^A F_A$ *and* $\xi \leq_s F_Q$ *both hold.*

Reason for using different directions for the orders in assumptions and guarantees is that "better" translates as $\geq_s^A$ for assumptions and $\leq_s$ for guarantees, see the beginning of section 3.3. To ensure soundness of any contract based approach for QoS, (probabilistic) monotonicity must be assumed, formalizing that, if any of the called services performs better, then as a result the orchestration should also perform better:

Assumption 2 (monotonicity) *The considered probabilistic OrchNet is* monotonic*, meaning that increasing the QoS guarantees for each transition t results in an increase in the end-to-end QoS of the OrchNet, and decreasing the assumptions on the OrchNet results in a decreasing of the assumptions for each transition t.*

A formal definition of probabilistic monotonicity regarding QoS was given in definitions 8 and 10 and conditions for monotonicity were stated in section 4. Even if conditions for monotonicity are not satisfied, by using theorem 1, it is always sound to base SLA or contracts on the loose end-to-end QoS defined in (10). This, of course, will result in being more pessimistic when composing contracts, *i.e.,* when inferring, from the contracts the orchestration has with the services it calls, the QoS it can offer to its clients.

As for monotonicity with respect to assumptions, we argue that, unlike for guaranteed QoS, it is a very natural condition — see the discussion at the end of section 2.2.

We are now ready to investigate contract composition. Roughly speaking, contract composition consists in inferring, from the contracts the orchestration has with its called services, the overall contract it can offer to its clients.

However, this rough statement needs to be refined: While it is true that the overall guarantee that the orchestration can offer to its clients can be deduced from those it gets from its sub-contractors, the converse holds for assumptions. For instance, the query throughput at each called service is deduced from the query throughput the orchestration is subject to. Also, other assumptions collected in parameter β are carried by tokens, and thus propagate inward the orchestration. Thus contract composition rather possesses a game theoretic flavour that we formalize next.

Contract composition is performed on a probabilistic OrchNet $\mathcal{N}$. For the rest of the section, each transition $t \in \mathcal{T}$ of $\mathcal{N}$ will represent a sub-contracted service, called by the orchestration. Contract composition proceeds as follows:

1. Initial conditions: They consist of F^0_A, the assumed distribution for the orchestration, and $F^0_{Q,t}$, the guaranteed distribution offered by each $t \in \mathcal{T}$.

2. Simulation Phase: it consists of the following steps:

a. Draw successive random calls to the orchestration according to distribution F^0_A; each call to the orchestration generates zero, one, or several calls to each transition t of the orchestration;

b. For every such call to a transition t:

   – Record the value α_t of the assumption of the call to t;
   – Draw random QoS increment ξ_t from distribution $F^0_{Q,t}$;
   – Compute the QoS parameters of $t^\bullet$ using competition policy and (6).

c. For each call to the orchestration, record the resulting end-to-end QoS $E(\mathcal{N})$.

Performing step b for the successive calls to t yields an empirical estimate $F^1_{A,t}$ for the actually occurring assumptions for t. Performing step c for the successive

calls to t yields an empirical estimate F^1_Q of the actual end-to-end QoS of the orchestration.

3. Negotiation Phase: At this point, two cases may occur:

For the *good case*, pair $(F^1_{A,t}, F^0_{Q,t})$ is a contract considered acceptable by every transition t. The orchestration can then propose the contract (F^0_A, F^1_Q) to its client and the procedure terminates at this step.

For the *bad case*, pair $(F^1_{A,t}, F^0_{Q,t})$ is a contract considered not acceptable by $t \in \mathcal{T}'$, for some $\mathcal{T}' \subseteq \mathcal{T}$. The guarantees may be too demanding considering the assumptions. In this case we will re-run the above iterative process with new inputs. We have two alternative approaches to do this, depending on which inputs we choose to update:

> In the first approach, we keep $F^0_{Q,t}$ unchanged for every transition t. Then, we update the assumed distribution F^0_A for the orchestration to a distribution F^1_A such that $F^1_A >^A_s F^0_A$, i.e., F^1_A is more favorable than F^0_A for the orchestration. When running the simulation phase using F^1_A instead of F^0_A, a new assumed distribution $F^2_{A,t}$ results for t that is more favorable than $F^1_{A,t}$ for transition t. Having sufficiently weakened F^1_A should then yield an assumed distribution $F^2_{A,t}$ such that $(F^2_{A,t}, F^0_{Q,t})$ is now considered acceptable by every transition t.
>
> In the second approach, we do not change the assumed distributions F^0_A and $F^1_{A,t}$, but we relax the guaranteed distribution $F^0_{Q,t}$ for every $t \in \mathcal{T}'$ to $F^1_{Q,t}$. The guaranteed distributions are relaxed till $(F^1_{A,t}, F^1_{Q,t})$ is an acceptable contract for every $t \in \mathcal{T}'$.

The convergence of this procedure is proved in appendix A.1.

6 Experiments

In our experiments we implement the contract composition procedure described in section 5. We use Orc to model the orchestration and to specify the QoS behaviour (contracts) of the services involved in the orchestration. Our tool for performing contract composition is built upon an interpreter of Orc in Java, developed by members of the Orc team at the University of Texas at Austin [11]. We perform our experiments on the CarOnLine example of Table 1.

Initial Conditions: As described in section 5, the contract composition procedure is carried out in iterative steps. Each iteration requires two inputs. The first input is the the initial assumption distribution F^0_A agreed between the orchestration and its clients. We take F^0_A to be the inter-query time distribution of the calls to orchestration by its clients. We use an exponential distribution to model this inter-query time and the rate parameter is taken to be 5 (*i.e.*, 5 requests/sec).

The second input is the set of guaranteed distributions $F^0_{Q,t}$ offered by each of the contracted services t - here t ranges over *GarageA, GarageB, AllCredit, AllCreditPlus, GoldInsure, InsurePlus* and *InsureAll*. Observe in Table 1 that most of the services affect more than one QoS parameter. We thus have a probability distribution for each of the QoS parameters that t affects, and we then take t's

guaranteed distribution $F^0_{Q,t}$ to be the product of these distributions. Doing this, we are assuming independence of the different QoS parameters that t affects. Also observe that all the services of CarOnLine affect the latency parameter d. The guaranteed distribution for d, for each of the services of CarOnLine were derived from measurements made by calling six freely available services over the web. These services were published on the online repository XMethods [17]. 20,000 calls were made to each of these services and their response times were measured. The cumulative distribution function of the response times measured are shown in figure 4. The guaranteed distribution for the other QoS parameters

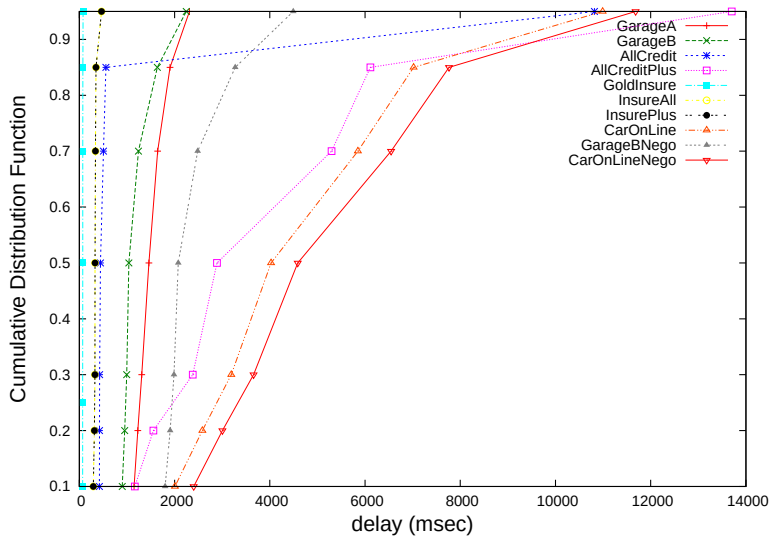

Figure 4: Cumulative distribution functions of latency, for the services of CarOnLine, and end-to-end latency. The result of the re-negotiation is shown in GarageBNego and CarOnLineNego.

affected by the services of CarOnLine is given in Table 2.

Simulation Phase: With the above initial conditions, we ran the simulation step with 100,000 calls to the orchestration. At each call, the inter-query time for each service was observed and the the end-to-end values of the QoS parameters d, l, c, p were observed. The resulting distribution for the assumptions $F^1_{A,t}$ - which is in fact the inter-query arrival time - for each of the services is given in Table 3. The orchestration's end-to-end QoS for the response time is shown by the CarOnLine curve in figure 4. The end-to-end values of the other QoS parameters are given in the 'O' columns in Table 2.

Negotiation Phase: Now suppose that for *GarageB*, the inter-query distribution $F^1_{A,t}$ is too strong for the guarantees $F^0_{Q,t}$ it offers. The contract with *GarageB* has to be re-negotiated. The first approach to re-negotiation is to relax the assumptions F^0_A which will reduce the assumptions $F^1_{A,t}$ at *GarageB*. If we adopt the second approach to re-negotiation, we keep the assumptions F^0_A and $F^1_{A,t}$ unchanged, but we sufficiently relax the guarantees of *GarageB*. The new guarantee for the response times of *GarageB* is given by the GarageBNego curve

l	GA	GB	O
0	0.25	0.2	0.21
1	0.25	0.25	0.25
2	0.25	0.25	0.25
3	0.15	0.25	0.23
4	0.1	0.05	0.06

p	GA	GB	O
10,000	0.15	0.18	0.17
12,000	0.15	0.14	0.14
15,000	0.2	0.23	0.23
20,000	0.25	0.15	0.16
25,000	0.25	0.3	0.3

i	GI	IP	IA	O
100	0.2	0.22	0.2	0.21
300	0.2	0.23	0.25	0.22
350	0.3	0.2	0.15	0.23
500	0.15	0.15	0.25	0.16
800	0.15	0.2	0.15	0.18

c	AC	ACP	O
1	0.25	0.2	0.2
2	0.2	0.2	0.2
3	0.15	0.25	0.24
4	0.2	0.15	0.16
5	0.2	0.2	0.2

Table 2: Probability distribution functions for the QoS parameters l, p, i and c of Table 1. The first column of each table gives the QoS parameter and its values. The other columns gives a service of CarOnLine, and the probability of getting a QoS value for that service. GA, GB, AC, ACP, GI, IP and IA are abbreviations for the services *GarageA, GarageB, AllCredit, AllCreditPlus, GoldInsure, InsurePlus* and *InsureAll* respectively. Columns labelled *O* represent the corresponding QoS value for the end-to-end orchestration.

Site Name	GarageA	GarageB	AllCredit	AllCreditPlus
Throughput	5.028	5.028	5.028	5.028

Site Name	GoldInsure	InsureAll	InsurePlus
Throughput	1.679	3.342	3.342

Table 3: Average throughput for each of the contracted sites.

in figure 4. The guarantees for the other QoS parameters are kept unchanged. The simulation step is now run again with this updated contract of *GarageB*. The corresponding guaranteed distribution for the orchestration is given by the CarOnLineNego curve in figure 4.

7 Conclusion

We have presented in this paper the foundations of a theory of QoS for Web service orchestrations. We adopt an approach based on contracts between the orchestrator and its sub-contracted services and its clients, having assumptions and guarantees on the QoS involved. For our approach to be sound, the orchestration needs to be monotonic with respect to the QoS parameters. We have proposed the use of probabilistic contracts, and have showed how they can be composed. We are currently improving our Java implementation to build a stable experimentation platform to support our studies.

References

[1] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[2] Samy Abbes and Albert Benveniste. True-concurrency probabilistic models: Branching cells and distributed probabilities for event structures. *Inf. Comput.*, 204(2):231–274, 2006.

[3] Rohit Aggarwal, Kunal Verma, John A. Miller, and William Milnor. Constraint driven web service composition in meteor-s. In *IEEE SCC*, pages 23–30, 2004.

[4] Jesús Arias-Fisteus, Luis Sánchez Fernández, and Carlos Delgado Kloos. Applying model checking to BPEL4WS business collaborations. In *SAC*, pages 826–830, 2005.

[5] Anne Bouillard, Sidney Rosario, Albert Benveniste, and Stefan Haar. Monotony in Service Orchestrations. In *Petri Nets*, 2009.

[6] Alexander Keller and Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *J. Network Syst. Manage.*, 11(1), 2003.

[7] Marco Ajmone Marsan, Gianfranco Balbo, Andrea Bobbio, Giovanni Chiola, Gianni Conte, and Aldo Cumani. The effect of execution policies on the semantics and analysis of stochastic petri nets. *IEEE Trans. Software Eng.*, 15(7):832–846, 1989.

[8] Jayadev Misra and William R. Cook. Computation orchestration: A basis for wide-area computing. *Journal of Software and Systems Modeling*, May, 2006. Available for download at `http://dx.doi.org/10.1007/s10270-006-0012-1`.

[9] T. Murata. Petri nets: Properties, analysis and applications. In *Proceedings of the IEEE*, volume 77, pages 541–580, April 1989.

[10] C. Ouyang, E. Verbeek, W.M.P van der Aalst, and S. Breutel. Formal Semantics and Analysis of Control Flow in WS-BPEL. BPM Center Report BPM-05-15, BPMcenter.org, 2005.

[11] Orc Language Project. `http://orc.csres.utexas.edu`.

[12] Sidney Rosario, Albert Benveniste, Stefan Haar, and Claude Jard. Probabilistic QoS and Soft Contracts for Transaction based Web Services. In *ICWS*, pages 126–133, 2007.

[13] Sidney Rosario, Albert Benveniste, Stefan Haar, and Claude Jard. Probabilistic QoS and Soft Contracts for Transaction based Web Services. *IEEE Transactions on Service Computing*, 1(4):1–14, oct–dec 2008.

[14] Sidney Rosario, Albert Benveniste, Stefan Haar, and Claude Jard. Flexible Probabilistic QoS Management of Transaction based Web Services orchestrations. In *ICWS*, 2009.

[15] S. Rosario and A. Benveniste and C. Jard. Shorter version of this paper, submitted for publication, 2009.

[16] T. Kamae and U. Krengel and G.L. O'Brien. Stochastic inequalities on partially ordered spaces. *The Annals of Probability*, 5(6):899–912, 1977.

[17] The XMethods Repository. http://www.xmethods.net.

[18] Wil M. P. van der Aalst. Verification of workflow nets. In *ICATPN*, pages 407–426, 1997.

[19] Liangzhao Zeng, Boualem Benatallah, Anne H. H. Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Trans. Software Eng.*, 30(5):311–327, 2004.

A Collecting proofs

A.1 Study of the contract composition procedure

To prove the convergence of this iterative procedure we will need the following assumptions:

Assumption 3

1. *For any contract* (F_A, F_Q)*, there exists a weaker contract* (F'_A, F'_Q)*, meaning that* $F'_A \leq_s^A F_A$ *and* $F'_Q \geq_s F_Q$*, that is acceptable to both parts engaged in the negotiation.*
2. *The considered probabilistic OrchNet is* strictly monotonic *w.r.t the assumptions, meaning that, for any transition* t *and any given* $F'_{A,t}$*, there exists a QoS assumption* F_A *for the orchestration that generates a QoS assumption* $F_{A,t}$ *for a transition* t *such that* $F_{A,t} \geq_s^A F'_{A,t}$*, i.e.,* $F_{A,t}$ *is better than* $F'_{A,t}$ *for transition* t*.*

Assumption 3.1 is "societal" because it is an assumption about the behavior of the "agents" that undertake contract negotiation. This assumption is not of a mathematical nature, unlike the second one, which is a strenghtening of monotonicity.

Proof of convergence of contract composition procedure: We successively study the first and the second approach for the negotiation phase.

For the first approach, observing convergence is simple. We observe that using assumption 3.1, we can find assumption distribution $F^2_{A,t}$ such that $(F^2_{A,t}, F^1_{Q,t})$ is an acceptable contract for t. Now using assumption 3.2, we can strengthen the assumptions F^1_A sufficiently enough, such that the assumptions that t is subject to is $F^2_{A,t}$.

For the second approach, use assumption 3.1 to find $F^1_{Q,t} \geq_s F^0_{Q,t}$ such that $(F^1_{A,t}, F^1_{Q,t})$ is now an acceptable contract for transition t. Now, re-running the simulation phase with initial conditions F^0_A and $F^1_{Q,t}$, yields $F^1_{A,t}$ as an assumption for each transition t, and F^2_Q as an updated guaranteed QoS for the orchestration. Since contract $(F^1_{A,t}, F^1_{Q,t})$ is acceptable to t, the orchestration can offer to its client F^2_Q as a guaranteed QoS. Observe that we do not need the stronger assumption 3.2 for the second approach.

A.2 Proof of Theorem 2

Proof: Throughout the proof, we fix an arbitrary value ω for the daemon. We first prove Statement 1. Let $\mathcal{N}' \in \mathbb{N}$ be such that $\mathcal{N}' \geq \mathcal{N}$. Since operators $\oplus$ and $\lhd$ are both monotonic, see definition 1, we have, by algorithm 2 and formulas (8) and (9):

$$E_\omega(\kappa(\mathcal{N}', \omega), \mathcal{N}') \geq E_\omega(\kappa(\mathcal{N}', \omega), \mathcal{N})$$

By (11) applied with $\kappa = \kappa(\mathcal{N}', \omega)$, we get that

$$E_\omega(\kappa(\mathcal{N}', \omega), \mathcal{N}) \geq E_\omega(\kappa(\mathcal{N}, \omega), \mathcal{N})$$

holds. This proves Statement 1. We prove statement 2 by contradiction. Let $(\mathcal{N}, \omega, \kappa^\dagger)$ be a triple violating Condition (11), in that

$$\kappa^\dagger \quad \text{cannot occur, but} \quad E_\omega(\kappa^\dagger, \mathcal{N}) \geq E_\omega(\kappa(\mathcal{N}, \omega), \mathcal{N}) \text{ does not hold.}$$

Now consider the OrchNet net $\mathcal{N}' = (N, \Phi, Q', Q_{\text{init}})$ where the family Q' is such that, $\forall t \in \kappa^\dagger$, $\xi'_t(\omega) = \xi_t(\omega)$ holds, and $\forall t \notin \kappa^\dagger$, using conditions 1 and 3 for operator $\oplus$ in definition 1 together with the assumption that $(\mathbb{D}, \leq)$ is an upper lattice, we can inductively select $\xi'_t(\omega)$ such that the following two inequalities hold:

$$\bigvee_{t \in \kappa^\dagger} q_t \quad \leq \quad (\bigvee_{p' \in {}^\bullet t} q_{p'}) \oplus \xi'_t(\omega) \tag{18}$$

$$\xi_t(\omega) \quad \leq \quad \xi'_t(\omega) \tag{19}$$

Condition (19) expresses that $\mathcal{N}' \geq \mathcal{N}$. By algorithm 1 defining competition policy, (18) implies that configuration $\kappa^\dagger$ can win all competitions arising in step 3 of competition policy, $\kappa(\mathcal{N}', \omega) = \kappa^\dagger$ holds, and thus

$$E_\omega(\kappa(\mathcal{N}', \omega), \mathcal{N}') = E_\omega(\kappa^\dagger, \mathcal{N}') = E_\omega(\kappa^\dagger, \mathcal{N})$$

However, $E_\omega(\kappa^\dagger, \mathcal{N}) \geq E_\omega(\kappa(\mathcal{N}, \omega), \mathcal{N})$ does not hold, which violates monotonicity. ◇

A.3 Proof of Theorem 3

Proof: Let φ_W be the net morphism mapping N_W onto W and let $\mathcal{N} \in \mathbb{N}$ be any OrchNet. We prove that condition 1 of Theorem 2 holds for $\mathcal{N}$ by induction on the number of transitions in the maximal configuration $\kappa(\mathcal{N}, \omega)$ that actually occurs. The base case is when it has only one transition. Clearly this transition has minimal QoS increment and any other maximal configuration has a greater end-to-end QoS value.

Induction Hypothesis. Condition 1 of Theorem 2 holds for any maximal occurring configuration with $m - 1$ transitions ($m > 1$). Formally, for a pre-OrchNet $\mathbb{N} = (N, \Phi, \mathbb{Q}, \mathbb{Q}_{\text{init}})$: $\forall \mathcal{N} \in \mathbb{N}, \forall \omega \in \Omega, \forall \kappa \in \mathcal{V}(N)$,

$$E_\omega(\kappa, \mathcal{N}) \geq E_\omega(\kappa(\mathcal{N}, \omega), \mathcal{N}) \tag{20}$$

must hold if $|\{t \in \kappa(\mathcal{N}, \omega)\}| \leq m - 1$.

Induction Argument. Consider the OrchNet $\mathcal{N}$, where the actually occurring configuration $\kappa(\mathcal{N}, \omega)$ has m transitions and let

$$\emptyset = \kappa_0(\omega) \subset \kappa_1(\omega) \cdots \subset \kappa_{M(\omega)}(\omega) = \kappa(\mathcal{N}, \omega)$$

be the increasing chain of configurations leading to $\kappa(\mathcal{N}, \omega)$ under competition policy, see (3.3). Let t be the unique transition such that $t \in \kappa_1(\omega)$. Let κ' be any other maximal configuration of $\mathcal{N}$. Then two cases can occur.

$t \in \kappa'$: In this case, comparing end-to-end QoS of $\kappa(\mathcal{N},\omega)$ and κ' reduces to comparing $E_\omega(\kappa(\mathcal{N},\omega)\setminus\{t\},\mathcal{N}^t)$ and $E_\omega(\kappa'\setminus\{t\},\mathcal{N}^t)$. Since $\kappa(\mathcal{N},\omega)\setminus\{t\}$ is the actually occurring configuration in the future $\mathcal{N}^t$ of transition t, using our induction hypothesis, then

$$E_\omega(\kappa'\setminus\{t\},\mathcal{N}^t) \geq E_\omega(\kappa(\mathcal{N},\omega)\setminus\{t\},\mathcal{N}^t)$$

holds, which implies

$$E_\omega(\kappa',\mathcal{N}) \;\geq\; E_\omega(\kappa(\mathcal{N},\omega),\mathcal{N})$$

$t \notin \kappa'$: Then there must exist another transition t' such that ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$. By the definition of clusters, $\varphi_W(t)$ and $\varphi_W(t')$ must belong to the same cluster **c**. Hence, $t^\bullet = t'^\bullet$ follows from condition 12 of Theorem 3. The futures $\mathcal{N}^t$ and $\mathcal{N}^{t'}$ thus have identical sets of transitions: they only differ in the initial marking of their places. If Q_{init} and Q'_{init} are the initial QoS values for the futures $\mathcal{N}^t$ and $\mathcal{N}^{t'}$, then $Q_{init} \leq Q'_{init}$ holds (since $\xi_t \leq \xi_{t'}$, $t^\bullet$ has QoS lesser than $t'^\bullet$ by monotonicity of $\oplus$). On the other hand,

$$E_\omega(\kappa(\mathcal{N},\omega),\mathcal{N}) \quad = \quad E_\omega(\kappa(\mathcal{N},\omega)\setminus\{t\},\mathcal{N}^t) \tag{21}$$

and

$$E_\omega(\kappa',\mathcal{N}) \;= E_\omega(\kappa'\setminus\{t'\},\mathcal{N}^{t'})$$

Now, since $\mathcal{N}^{t'}$ and $\mathcal{N}^t$ possess identical underlying nets and $\mathcal{N}^{t'} \geq \mathcal{N}^t$, then we get

$$E_\omega(\kappa'\setminus\{t'\},\mathcal{N}^{t'}) \quad \geq \quad E_\omega(\kappa'\setminus\{t'\},\mathcal{N}^t) \tag{22}$$

Finally, the induction hypothesis on (21) and (22) together imply $E_\omega(\kappa',\mathcal{N}) \geq E_\omega(\kappa(\mathcal{N},\omega),\mathcal{N})$.

This proves that condition 1 of Theorem 2 holds and finishes the proof of the theorem. ◇

A.4 Proof of Theorem 4

Proof: We will show that when condition (12) of Theorem 3 is not satisfied by W, the Orchnets in its induced preOrchNet $\mathbb{N}_W$ can violate condition (11) of Theorem 2, the necessary condition for monotonicity.

Let c_W be any cluster in W that violates the condition 12 of Theorem 3. Consider the unfolding of W, N_W and the associated morphism $\varphi : N_W \mapsto W$ as introduced before. Since W is sound, all transitions in c_W are reachable from the initial place i and so there is a cluster c in N_W such that $\varphi(c) = c_W$. There are transitions $t_1, t_2 \in c$ such that ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, ${}^\bullet\varphi(t_1) \cap {}^\bullet\varphi(t_2) \neq \emptyset$ and $\varphi(t_1)^\bullet \neq \varphi(t_2)^\bullet$. Call $[t] = \lceil t \rceil \setminus \{t\}$ and define $\kappa = [t_1] \cup [t_2]$. We consider the following two cases:

κ is a configuration. If so, consider the OrchNet $\mathcal{N}^* \in \mathbb{N}_W$ obtained when transitions of N_W (and so W) have QoS increments as that in $\mathcal{W}^*$. So for the daemon value ω^*, the quantity $E_{\omega^*}(\kappa, \mathcal{N}^*)$ is some finite value $\mathbf{q}^*$. Now, configuration κ can actually occur in a OrchNet $\mathcal{N}$, such that $\mathcal{N} > \mathcal{N}^*$, where $\mathcal{N}$ is obtained as follows (ξ and ξ^* denote the QoS increments of transitions in $\mathcal{N}$ and $\mathcal{N}^*$ respectively): $\forall t \in \kappa, t' \in N_W$ s.t. ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$, using additional condition (13) of Theorem 4, select $\xi_{t'}(\omega^*)$ such that $\xi_{t'}(\omega^*) > \mathbf{q}^*$ and keep the other QoS increments unchanged. In this case, for the daemon value ω^*, the QoS increments of all transitions of $\mathcal{N}$ (and so its overall execution time) is finite. Denote by $\mathcal{N}^\kappa$ the future of $\mathcal{N}$ once configuration κ has actually occurred. Both t_1 and t_2 are minimal and enabled in $\mathcal{N}^\kappa$.

Since $\varphi(t_1)^\bullet \neq \varphi(t_2)^\bullet$, without loss of generality, we assume that there is a place $p \in t_1^\bullet$ such that $\varphi(p) \in \varphi(t_1)^\bullet$ but $\varphi(p) \notin \varphi(t_2)^\bullet$. Let t^* be a transition in $\mathcal{N}^\kappa$ such that $t^* \in p^\bullet$. Such a transition must exist since p can not be a maximal place: $\varphi(p)$ can not be a maximal place in W which has a unique maximal place. Now, consider the Orchnet $\mathcal{N}' > \mathcal{N}$ obtained as follows: using repeatedly condition 3 for operator $\oplus$ in definition 1, $\xi'_{t_1}(\omega^*) = \xi_{t_1}(\omega^*), \xi'_{t_2}(\omega^*) \geq \xi_{t_1}(\omega^*)$, and, for all other $t \in c, \xi'_t(\omega^*) \geq \xi'_{t_2}(\omega^*)$. For all remaining transitions of $\mathcal{N}'$, with the exception of t^*, the QoS increments are the same as that in $\mathcal{N}$ and thus are finite for ω^*. Finally, select $\xi'_{t^*}(\omega^*)$ such that

$$\xi_{t_1}(\omega^*) \oplus \xi'_{t^*}(\omega^*) > \mathbf{Q}^* \tag{23}$$

where $\mathbf{Q}^* \in \mathbb{D}$ will be chosen later — here we used the special condition (13) on $\mathbb{D}$ together with condition 3 for operator $\oplus$ in definition 1. Transition t_1 has a minimal QoS increment among all transitions in c. It can therefore win the competition, thus giving raise to an actually occuring configuration $\kappa(\mathcal{N}', \omega^*)$. Select $\mathbf{Q}^*$ equal to the maximal value of the end-to-end QoS of the set K of all maximal configurations κ that do not include t_1 (for eg, when t_2 fires instead of t_1). By (23), since t^* is in the future of t_1, we thus have $E_{\omega^*}(\kappa(\mathcal{N}', \omega^*), \mathcal{N}') \geq \xi_{t_1}(\omega^*) \oplus \xi'_{t^*}(\omega^*) > \mathbf{Q}^* \geq E_{\omega^*}(\kappa, \mathcal{N}')$ for any $\kappa \in K$ and so $\mathcal{N}'$ violates the condition (11) of Theorem 2.

κ is not a configuration. If so, there exist transitions $t \in [t_1] \setminus [t_2]$, $t' \in [t_2] \setminus [t_1]$ such that ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$, ${}^\bullet \varphi(t) \cap {}^\bullet \varphi(t') \neq \emptyset$ and $\varphi(t)^\bullet \neq \varphi(t')^\bullet$. The final condition holds since t_2 and t_1 are not in the causal future of t and t' respectively. Thus t and t' belong to the same cluster, which violates condition 12 of Theorem 3 and we can apply the same reasoning as in the beginning of the proof. Since $[t]$ is finite for any transition t, we will eventually end up with κ being a configuration.

$\diamond$

A.5 Proof of theorem 7

Proof: The proof is by contradiction. Assume that $\mathbb{N}$ is *not* probabilistically monotonic. By theorem 5, this implies that $\mathbb{N}$ is *not* monotonic with positive

P-probability, i.e., :

$$\begin{array}{l}\text{there exists a pair } (\mathcal{N},\mathcal{N}') \text{ of OrchNets such that}\\ \mathcal{N} \geq \mathcal{N}' \text{ and } \mathbf{P}\,\{\,\omega \in \Omega \mid E_\omega(\mathcal{N}) < E_\omega(\mathcal{N}')\} > 0.\end{array} \tag{24}$$

To prove the theorem it is enough to prove that (24) implies:

$$\begin{array}{l}\text{there exists } \mathcal{N}_o, \mathcal{N}'_o \in \mathbb{N} \text{ such that } \mathcal{N}_o \geq \mathcal{N}'_o,\\ \text{but } E(\mathcal{N}_o) \geq_s E(\mathcal{N}'_o) \text{ does not hold}\end{array} \tag{25}$$

To this end, set $\mathcal{N}_o = \mathcal{N}$ and define $\mathcal{N}'_o$ as follows, where Ω_o denotes the set $\{\,\omega \in \Omega \mid E_\omega(\mathcal{N}) < E_\omega(\mathcal{N}')\}$:

$$\mathcal{N}'_o(\omega) = \text{ if } \omega \in \Omega_o \text{ then } \mathcal{N}'(\omega) \text{ else } \mathcal{N}(\omega)$$

Note that $\mathcal{N}_o \geq \mathcal{N}'_o$ by construction. Also, $\mathcal{N}'_o \geq \mathcal{N}'$, whence $\mathcal{N}'_o \in \mathbb{N}$ since condition 2b of theorem 2 is satisfied. On the other hand, we have $E_\omega(\mathcal{N}_o) < E_\omega(\mathcal{N}'_o)$ for $\omega \in \Omega_o$, and $E_\omega(\mathcal{N}_o) = E_\omega(\mathcal{N}'_o)$ for $\omega \notin \Omega_o$. By (24), we have $\mathbf{P}(\Omega_o) > 0$. Consequently, we get:

$$[\,\forall \omega \in \Omega \;\Rightarrow\; E_\omega(\mathcal{N}_o) \leq E_\omega(\mathcal{N}'_o)\,] \quad \text{and} \quad [\,\mathbf{P}\,\{\,\omega \in \Omega \mid E_\omega(\mathcal{N}_o) < E_\omega(\mathcal{N}'_o)\} > 0\,]$$

which implies that $E(\mathcal{N}_o) \geq_s E(\mathcal{N}'_o)$ does not hold. ◇

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)

ISSN 0249-6399