Probabilistic QoS and soft contracts for transaction based Web services orchestrations

Sidney Rosario
IRISA/INRIA
Rennes, France
srosario@irisa.fr

Albert Benveniste
IRISA/INRIA
Rennes, France
benveniste@irisa.fr

Stefan Haar
SITE, University of Ottawa, Canada
shaar@site.uottawa.ca

Claude Jard
ENS Cachan Bretagne, France
jard@bretagne.ens-cachan.fr

***Abstract*— Web services orchestrations and choreographies require establishing Quality of Service (QoS) contracts with the user. This is achieved by performing QoS composition, based on contracts established between the orchestration and the called Web services. These contracts are typically stated in the form of hard guarantees (e.g., response time always less than 5 msec).**

In this paper we propose using *soft contracts* instead. Soft contracts are characterized by means of probability distributions for QoS parameters. We show how to compose such contracts, to yield a global contract (probabilistic) for the orchestration. Our approach is implemented by the *TOrQuE* tool. Experiments on *TOrQuE* show that overly pessimistic contracts can be avoided and significant room for safe overbooking exists.

I. Introduction

Web Services Orchestrations have attracted growing interest over the last years [4], [13]. They are now considered an infrastructure of choice for managing business processes and workflow activities over the Web infrastructure [2]. In this context, the Web services for composition are mainly of transactional nature.

BPEL [4] has become the industrial standard for specifying orchestrations. Numerous studies have been devoted to relating BPEL to mathematical formalisms for workflows, such as WorkFlow nets (WFnets) [5] a special subclass of Petri nets, or the pi-calculus [7]. This has allowed developing analysis techniques and tools for BPEL [8], [10] including functional aspects of contracts [6], as well as techniques for workflow mining from logs [9].

When applied to the management of OEM/supplier cooperations, orchestrations must make precise the duties and responsibilities of the different actors in such chains. This is achieved by relying on *contracts* [3]. Contracts specify the requirements each subcontractor must satisfy; from these, the overall contract between orchestration and its customers can be established. This process is called *contract composition.*

While functional aspects of contract composition rely on the above mentioned formal models and techniques [6], *Quality of Service (QoS)* aspects must be handled as well. The Web Service Level Agreement (WSLA) framework [12] is a standard proposed by IBM for specifying (and monitoring) QoS parameters in Web Services. There are no predefined QoS parameters in WSLA, the contracting parties are free to negotiate and define their own QoS parameters in a flexible manner. This flexibility is essential because different organisations often associate different semantics to the same parameter name.

Nevertheless, most SLAs commonly tend to have QoS parameters which are mild variations of the the following measures:

- response time (latency);
- availability;
- maximum allowed query rate (throughput);
- security.

In this paper we do not consider aspects of security in QoS.

To the best of our knowledge, with the noticeable exception of [20], all composition studies consider performance related QoS parameters of contracts in the form of *hard bounds.* For instance, response times and query throughput are required to be less than a certain fixed value and validity of answers to queries must be guaranteed at all times. When composing contracts, hard composition rules are used such as addition or maximum (for response times), or conjunction (for validity of answers to queries).

Whereas this results in elegant and simple composition rules, we argue that this general approach by using hard bounds does not fit the reality well. Figure 1 displays a

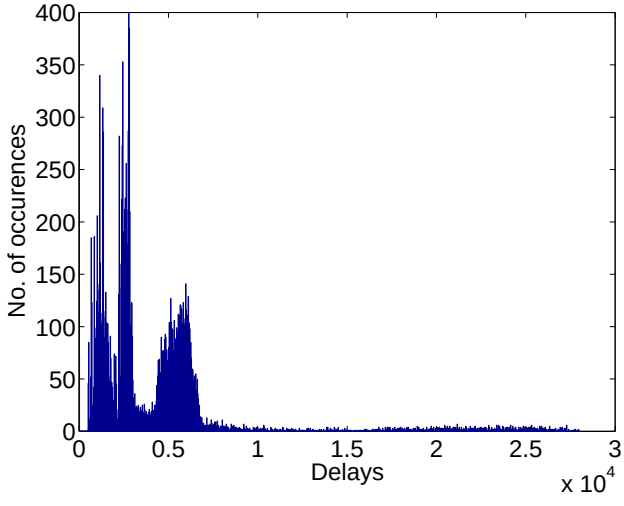

Fig. 1. Measurement records for response times, for Web service `StockQuote`.

Research funded by: French national telecommunication research network *(RNRT)* project *SWAN*, decision no. 03 S 481

histogram of measured response times for a "StockQuote" Web Service which returns stock prices of a queried entity [29]. These measurements show evidence that the tail of the above distribution cannot be neglected. For example, in this histogram, quantiles of 90%, 95%, and 98%, correspond to response times of 6,494 ms, 13,794 ms, and 23,506 ms respectively. Setting hard bounds in terms of response time would amount to selecting, e.g., the 98% quantile of 23,506 ms, leading to an over pessimistic promise, for this service.

In fact, users would find it very natural to "soften" contracts: a contract should promise, e.g., a response time in less than T milli-sec for 95% of the cases, validity in 99% of the cases, accept a throughput not larger than N queries per second for 98% of a time period of M hours, etc. This sounds reasonable but is not used in practice, partly because soft contracts based on quantiles as above are not supported by composition rules.

In this paper we propose a fully probabilistic approach to soft QoS contract composition. We advocate that soft contracts should be based on probability distributions, not on quantiles, for the following reasons. Contracts expressed as quantiles do not compose as such. In contrast, composing probability distributions is simply achieved by running Monte-Carlo simulations. Assume a combined executable functional-and-QoS model of the orchestration is available. Then, drawing QoS parameters for the called sites in accordance with their probabilistic contracts and feeding Monte-Carlo simulations with these, yields an estimate of the probability distribution for the different QoS parameters of the orchestration. Thus, while probabilistic contracts seem, at a first glance, technically involved, they compose easily. This is a major advantage when considering contract composition.

Such a combined executable functional-and-QoS model of the orchestration can be obtained by enhancing orchestration specifications with QoS attributes seen as random variables, and then combining them properly. This requires exhibiting concurrency and sequentialization in the orchestration in a precise way, which amounts to representing orchestrations as *partial orders* of events. Mathematical models of orchestrations typically provide this. For example, the partial order semantics of WFnets [5] is such a mathematical model. Our group has developed a tool *TOrQuE* (**T**ool for **Or**chestration **Qu**ality of Service **e**valuation) that directly produces executions as partial orders, from an ORC program. ORC is a simple and clean academic language for orchestrations with a rigorous mathematical semantics, developed by Cook and Misra at Austin [23]. The Monte-Carlo results reported here were obtained by this tool. However, it should be clear that our approach of soft probabilistic contracts applies to any orchestration formalism as soon as the orchestration can be animated as a partial order of events.

Using these tools, we show in particular that the orchestration QoS parameters obtained by Monte-Carlo simulations using probabilistic contracts are much less pessimistic than those based on hard bound reasoning, thus showing evidence of opportunities for well sound overbooking.

The paper is organized as follows. Section II gives a survey of the existing literature on QoS-enabled WS composition. In section III, we present our general approach and the *TOrQuE* tool supporting it. In section IV, we introduce the running example used throughout (an orchestrated service for buying cars online), and we present simulation results, exhibiting the potential for overbooking. Finally, section V presents conclusions and outlooks.

II. Related Work

In this section we survey the work done on QoS-based Web Service composition. Proposals for such QoS-based compositions are few and no well-accepted standard exists to date. The METEOR-S project [14] has studied QoS enabled compositions in workflows. In METEOR-S, Agarwal et. al [15] view QoS based composition as a constraint satisfaction/optimization problem. Services have selection criteria which are constraints, for which an optimal solution is found using integer linear programming. Cardoso et al. in [16] aim to derive QoS parameters for a workflow, given the QoS parameters of its component tasks. Using a graph reduction technique, they repeatedly re-write the workflow, merging different component tasks and also their QoS attributes according to different rules.

Zeng et al. [17] use Statecharts to model composite services. An orchestration is taken to be a finite execution path. For each task of the orchestration, a service is selected from a pool of candidate services, using linear programming techniques such that it optimizes a specific global QoS criteria. In [18], the authors propose using fuzzy distributed constraint satisfaction programming (CSP) techniques for finding the optimal composite service.

Canfora et. al [19] suggest using Genetic Algorithms for deriving optimal QoS compositions. They use techniques similar to [16] for modeling QoS of services. Compared to the linear programming method of Cardoso et. al [16], the genetic algorithm is typically slower on small to moderate size applications, but is more scalable, outperforming linear programming techniques when the number of candidate services increase.

A distinguishing feature of our proposal from the above composition techniques is that we do not consider the QoS parameters of a service to be fixed, hard bound values. We believe that in reality, these parameters exhibit significant variations in their values and are better modeled by a probability distribution. This alternative approach has two advantages. First, it reduces pessimism in contract composition, as we shall see. And, second, it allows for "soft" monitoring of contract breaching (have a delayed response once upon a time should not be seen as a breaching). We propose using simulation techniques to analyse the QoS of a composite service.

III. General approach and the *TOrQuE* tool

A. General approach

The Open World paradigm: For QoS analysis, actors for consideration are:

- the orchestration;

- the Web services called by the orchestration;
- the transport network infrastructure.

All these actors contribute to the overall QoS characteristics of the orchestration. Therefore, to be able to offer QoS guarantees, the orchestration needs QoS data from the other two types of actors.

In the context of networks, QoS studies assume knowledge of end-to-end resources and traffic, and use these to predict or estimate end-to-end QoS [11]. This can, for example, be used for evaluating the end-to-end performance of streaming services, supported by a dedicated cross-domain VPN. Once defined and deployed, the considered VPN has knowledge of his resources and traffic. For our case, the situation, however, is different:

- The orchestration has direct knowledge of the resources of its own server architecture. It knows the traffic it can support, and it can monitor and measure the ongoing traffic at a given time.
- The resources and extra traffic for each called Web service is not known to the orchestration—other users of these sites belong to the "open world" and the orchestration just ignores their existence.
- The resources and extra traffic for the transport network infrastructure are not known to the orchestration—other traffic belongs to the "open world" and the orchestration just ignores it.

Contracts have therefore emerged as the adequate paradigm for QoS of orchestrations and, more generally, of composite Web services in open world contexts.

Contracts: A *contract* consists of an agreement on QoS parameters of the kind listed before. Contracts provide the orchestration with the information needed to construct its own offer to customers.

In all cases, the orchestration will not contract regarding transport. The reason is that the orchestration does not want to know the network domains it may traverse. If, however, QoS information regarding the transport layer is wanted, this can be coarsely estimated by sending "pings" to the considered site. On the other hand the orchestration may contract with the Web services it is calling—some sites, however, such as e.g., Google, may be targeted to huge sets of users and would therefore not enter in a negotiation process with any orchestration. Therefore, in designing its contracts with its own customers, the orchestration: 1/ uses the contracts it has agreed upon with its subcontracting Web services, 2/ may estimate QoS parameters for other Web services it is using, and, 3/ may estimate QoS parameters for transport.

Classically, contracts are formulated as hard bounds on some QoS parameters. As argued in the introduction, it is preferable to characterize contracts in terms of probability distributions over QoS parameters. Hard bounds on parameters will then be replaced by "soft" bounds, of probabilistic or statistical nature.

Contract composition: From the above discussion, the need for *probabilistic contract composition* emerges. We have developed the following Monte-Carlo technique for QoS contract composition to be performed at design time:

- Contracts with the called sites have the form of probability distributions for the considered QoS parameters. From these, we draw successive outcomes for the tuples:

  $$\{\text{response to queries, associated QoS parameters}\}$$

  If no contract is available for a given site, we replace the missing probability distribution by empirical estimates of it, based on QoS measurements.
- Using a partial order execution model for the orchestration, we run QoS-enhanced Monte-Carlo simulations of the orchestration, thus deriving empirical estimates for the global QoS parameters of the orchestration.
- Having these empirical estimates, we can properly select quantiles defining soft contracts for the end user.

B. The TOrQuE tool

The *TOrQuE* (***T**ool for **Or**chestration simulation and **Qu**ality of service **E**valuation*) tool implements the above methodology. Its overall architecture is shown in Figure 2. The steps involved in the QoS evaluation and the *TOrQuE*

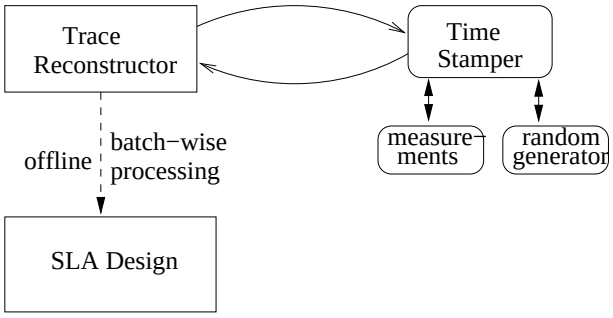

Fig. 2. Overall architecture of the *TOrQuE* tool.

modules that perform them are commented next.

1) The orchestration model: To ease the development of this tool, we decided to replace the (complex) BPEL standard for specifying Web services orchestrations by a light weight formalism called ORC [23]. ORC has a rigorous mathematical semantics based on SOS rules. The authors of this formalism have developed a tool [24] which can animate orchestrations specified in ORC.

2) Getting QoS enhanced partial order models of executions with the "Trace Reconstructor": Jointly with the authors of ORC, we have developed an alternative mathematical semantics for ORC in terms of *event structures* [25]. Event structures [21] provide the adequate paradigm for deriving partial order models of ORC executions, in which causality and concurrency relationships between the different events of the orchestration is made explicit. Partially ordered executions can be tagged with QoS parameters which can then be composed. For example, figure 3 shows how the response time of a fork-join pattern is computed from that of its individual events. These max-plus rules are used to combine delays in the partial order. The QoS parameter

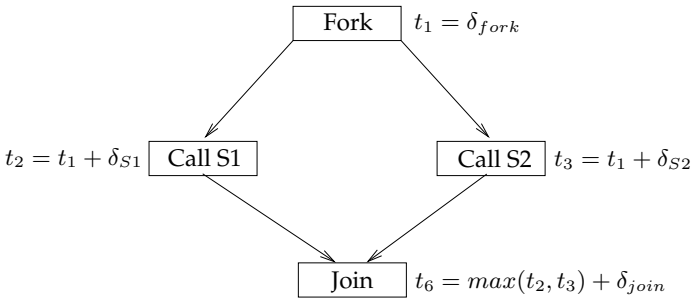

Fig. 3. Deriving response time for a fork-join pattern. The "Fork" and "Join" are the branching and synchronization events, $S1$ and $S2$ are two Web Services called in parallel. δ_a denotes the time taken for event a to execute.

tagging of the partial ordered executions and their composition is implemented in *TOrQuE*'s *trace reconstructor* module (see figure 2). Arbitrary patterns encountered in ORC specifications can be handled by this module.

3) Drawing at random, samples of QoS parameters for the called sites, with the "Time Stamper": To perform Monte-Carlo simulations using the Trace Reconstructor, we need to feed it with actual values for the QoS parameters. For the called sites, these values should be representative of the contracts established between them and the orchestration. This is achieved by drawing such parameters at random from the probability distribution specified in each contract.

If no contract is available with a given site, the needed probability distribution may alternatively be estimated from measurements. For example, calling the considered site a certain number of times and recording the response times provides an empirical distribution that can be re-sampled by simple bootstrapping techniques [22]. The Time Stamper module supports both techniques: sampling from contract's probability distribution or bootstrapping measured values.

4) Exploiting results from Monte-Carlo simulations to set contracts for the orchestration with the "SLA Design Unit": This is mainly a GUI module that displays simulation logs and histograms or empirical distributions of the QoS parameters and allows selecting appropriate quantiles.

IV. RESULTS: OPPORTUNITIES FOR OVERBOOKING

In this section we report the results obtained from the simulations of the *TOrQuE* tool which validate our approach of using probabilistic contracts.

A. Illustrative example

We perform our experiments on the `CarOnLine` example developed in the *SWAN* project [30]. `CarOnLine` is a composite service for buying cars online, together with credit and insurance. A simplified schematic description of the service is given in figure 4.

On receiving a car model as an input query, the CarOnLine service first sends parallel requests to two car dealers (`GarageA,GarageB`), getting quotations for the car. We guard the calls to each garage by a timer, which kills the waiting when timeout occurs. The best offer (minimum price)

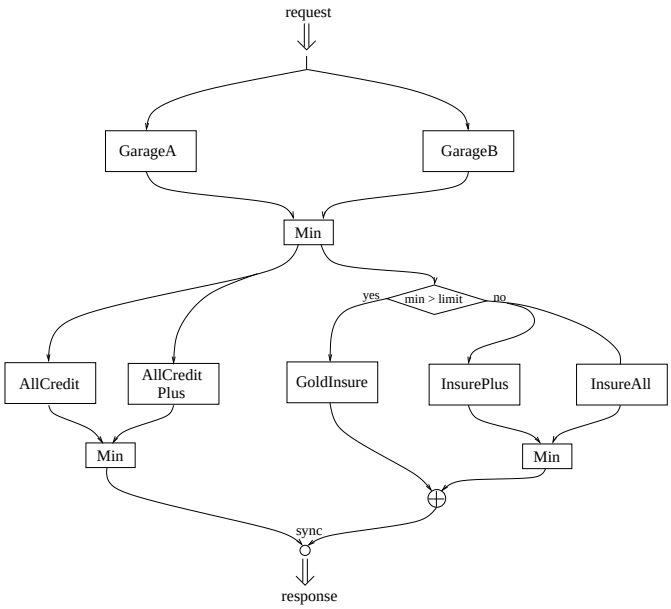

Fig. 4. A simplified view of the CarOnLine orchestration. The calls to GarageA and GarageB are guarded by a timer that returns a "Fault" message whenever the timeout occurs.

is selected and credit and insurances are parallelly found for the offer. Two banks (`AllCredit,AllCreditPlus`) are queried for credit rates and the one offering a lower rate is chosen. For insurance, if the car price of the best offer is greater than a certain limit, any insurance offer by service `GoldInsure` is accepted. If not, two services (`InsurePlus,InsureAll`) are parallelly called and the one offering the lower insurance rate is chosen. In the end, the (`car-price,credit-rate,insurance-rate`) tuple is returned to the requester.

The ORC program for CarOnLine is given in table I. ORC defines three basic 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. "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 of g is terminated. All site calls in f having x as a parameter are blocked till x is defined (*i.e.*, till g returns its first value).

CarPrice parallelly calls *GarageA* and *GarageB* for quotations. Calls to these garages are guarded by a timer site *Timer* which returns a fault value T time units after the calls are made. The *let* site simply returns the values of its arguments—sites can only execute when all their parameters are defined and thus can be used to synchronize parallel threads. The value returned by *CarPrice* (here the variable p) is passed as argument to *GetCredit* and *GetInsur* which parallelly find credit and insurance rates for the price.

Figure 5 shows a diagram of the event structure corresponding to the `CarOnLine` program written in ORC. The event structure is generated by our tool and it collects all the possible executions of `CarOnLine`, taking into account

$$
\begin{array}{lll}
CarOnline(car) & \triangleq & CarPrice(car) \quad >p> \\
 & & let(p,c,r) \\
 & & \quad \textbf{where} \;\; c :\in GetCredit(p) \\
 & & \qquad\quad r :\in GetInsur(p)
\end{array}
$$

$$
\begin{array}{lll}
CarPrice(car) & \triangleq & \{Mux(p1,p2) \\
 & & \quad \textbf{where} \\
 & & \qquad p1 :\in GarageA(car) \mid Timer(T) \\
 & & \qquad p2 :\in GarageB(car) \mid Timer(T) \\
 & & \} \\
 & & \quad >p> \;\; if(p \neq Fault)) \gg let(p)
\end{array}
$$

$$
\begin{array}{lll}
GetCredit(p) & \triangleq & Min(r1,r2) \\
 & & \quad \textbf{where} \\
 & & \qquad r1 :\in AllCredit(p) \\
 & & \qquad r2 :\in AllCreditPlus(p)
\end{array}
$$

$$
\begin{array}{lll}
GetInsur(p) & \triangleq & \{if(p \geq limit) \gg GoldInsure(p)\} \\
 & & \qquad\qquad \mid \\
 & & \{if(p \leq limit) \gg \\
 & & \qquad min(ip,ia) \\
 & & \qquad\quad \textbf{where} \\
 & & \qquad\qquad ip :\in InsurePlus(p) \\
 & & \qquad\qquad ia :\in InsureAll(p) \\
 & & \}
\end{array}
$$

TABLE I
CARONLINE IN ORC.

timers and other interactions between data and control. Each execution has the form of a partial order and can be analysed to derive appropriate QoS parameter composition, for each occurring pattern. Each site call to a service M is translated into three events, the *call* (M), the *call return* ($?M$) and the *publish action* (!), which adds to the length of the structure. For more information regarding these event structures, the reader is referred to [25].

In orchestrations, exceptions and their handling are frequently part of the orchestration specification itself. In addition, collecting measurement data from existing Web services regarding this type of parameter is difficult (actually, in our experiments, no exceptions were observed). For these two reasons, we did not include exceptions in our simulation study.

B. Analyzing Response Times: approach

Probabilistic contracts for the sites: The sites in the CarOnLine example were not implemented as real services over the internet. In order to assign realistic delay behaviour to these sites during the simulations, we associated their behaviour to that of actual Web Services over the internet. For this, we measured response times of calls to these actual Web Services. The response time recorded were used in a bootstrap mode and also to fit distributions which would be sampled during simulations.

We considered six different Web Services for this purpose [29]: `StockQuote` which returns stock prices for a queried enterprise, `USWeather` which gives the weather forecast of a queried city for a week from the day of the call, `CongressMember` which returns the list of the members of the US Congress, `Bushism` which returns a random quote of George W. Bush, `Caribbean` which returns information related to tourism in the Caribbean, and `XMethods` which queries a database of existing Web Services over the web.

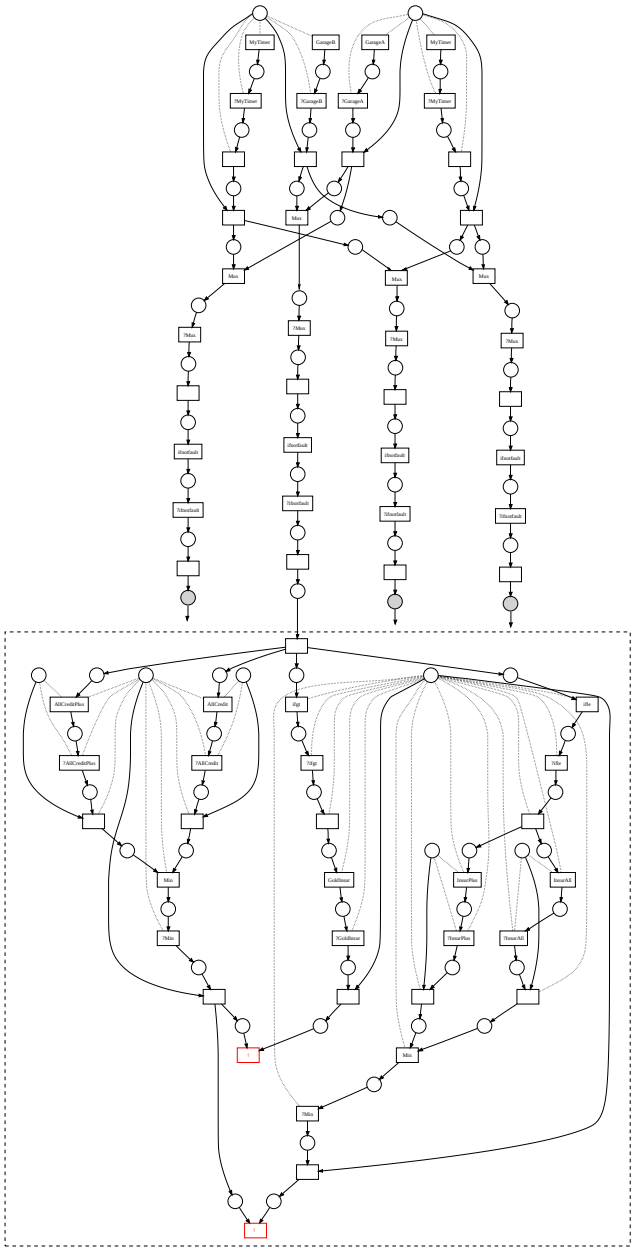

Fig. 5. A labelled event structure collecting all possible executions of `CarOnLine`, as generated by our tool. The three dangling arcs from the shaded places are followed by copies of the boxed net.

We made 20,000 calls to each of these six Web Services and measured their response times. The calls were made in sequence, a new call being made as soon as the previous call responded. We could roughly categorise these services into three categories based on their response times:

- Fast: The service *Caribbean* with response times in the range 60-100 ms or the *CongressMember* service with response times between 300-500 ms.
- Slow : Service *StockQuote* which responded typically between 2 and 8 seconds.
- Moderate: The services like *USWeather*, *XMethods* and *Bushism*, with response times in the 800-2000 ms range.

Fitting distributions on measured data: To validate the use of certain families of distributions, we performed their best fit on the measured data. When applied to the measured response times of the six different web services, we observed that T location-scale distributions served as good approximations in most cases. Moreover, Gamma and the Log-Logistic distributions were also reasonably good fits for the response times. Figure 6 shows the results of the fit of a T Location-Scale distribution on the response times of the service `USWeather`.

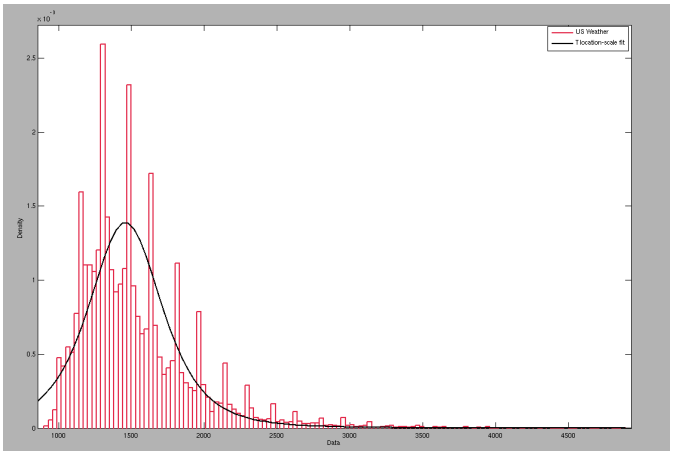

Fig. 6. Fitting of a T Location-scale distribution on the plot of 20,000 measured delays of the service *USWeather*.

While the quality of fit is reasonably good, this point is anyway not central in our study. We only see the use of certain families of distributions as an alternative to bootstrap techniques, when measurements are not available. In general, however, we prefer using bootstrapping techniques.

Orchestration Engine Overhead: The events of an orchestration could be seen as one of these two types : 1/ the service call events which are calls to a external sites. 2/ the events internal to the orchestration, implementing the processing and coordination actions of the orchestration. Depending on the relative cost (in terms of execution time) of these events the following scenarios can be considered:

- Zero delay: The delay due to the internal events is zero (or negligible) when compared to that of the site calls. The overall delay of the orchestration would depend solely on the response times of the services it calls in this case.
- Non-zero delay: The delays of the internal events in this case are non zero, comparable to the delays of site calls.

Since the performance of our prototype can not be regarded as representative of that of a real orchestration engine, we considered only the first scenario.

C. Simulation results

All the measurements and simulations were performed on a 2 GHz Pentium dual core processor with 2 Gb RAM. We consider two cases of simulations, depending on the timeout value T for the calls to the garages (see site $Timer(T)$ in table I) : 1) No timeout (equally, T is infinite) 2) T is a finite value, which is lesser than the maximum response time of a garage.

Case 1: No timeouts

Based on the way delays of site calls are generated, we performed two types of simulations: those in which delays generation is done by 1/ bootstrapping measured values, 2/ sampling a T location-scale distribution, previously fit to measured data.

1) Bootstrap based Simulations: In these simulations, we associated each service in the CarOnLine example with delay behaviours of one of the six web services mentioned previously. The associations are shown in Table II.

Site	Service
GarageA	USWeather
GarageB	Bushism
AllCredit	XMethods
AllCreditPlus	StockQuote
GoldInsure	Caribbean
InsureAll	CongressMembers
InsurePlus	CongressMembers

TABLE II
RESPONSE TIME ASSOCIATIONS FOR SITES IN CARONLINE

The service "Min" in `CarOnLine` (figure 4) is considered to be a local service at the orchestrator with negligible response times. During any run of CarOnLine, the response time of a call is picked uniformly from the set of 20,000 delay values of its associated site.

Results using hard contracts: Consider the following "hard contract" policy—which is close to current state of practice. Contracts have the form of a certain quantile, e.g.: "the response time shall not exceed x ms in y% of the cases."

More precisely, let contracts of the orchestration with a site be of the form

$$\mathbb{P}(\delta_i < K_i) \quad \geq \quad p_i \tag{1}$$

where $i = 1, ..., m$ ranges over the sites involved in the orchestration, δ_i is the response time of site i, K_i is the promised bound of site i, and p_i is the corresponding probability (so that $\delta_i < K_i$ holds in $y\%$ of the cases, where $y = 100 \times p_i$). Assuming the called sites to be probabilistically independent, what the orchestration can guarantee to its clients is

$$\mathbb{P}(\delta < K) \quad \geq \quad \prod_{i=i}^{m} p_i \tag{2}$$

where δ is the response time of the orchestration and K is the max-plus combination the K_i's, according to the orchestration's partial ordering of call events.

By setting the delay contracts (maximum delay values) of each of the sites involved in CarOnLine to their 99.2% quantile values, we get the end-to-end orchestration delay

bound to be $K = 66,010$ ms, which can be guaranteed for 94.53% of the cases.

Recall that, if only the quantile (*i.e.*, the pair (K_i, p_i)) is known as part of the contract with each called site, then quantile (2) cannot be computed. The reason is that quantiles do not compose.

Results using probabilistic soft contracts: We now compare the above results with our approach using probabilistic contracts. To this end, we performed 100,000 runs of the orchestration in the bootstrap mode. The empirical distribution of end-to-end delays of the orchestration is shown in figure 7. The minimum delay observed in this case is 1511 ms and the maximum is 369559 ms. The 94.53% delay quantile of this distribution is 23,189 ms, to be compared with the more pessimistic value 44,243 of ms that we obtained using the usual approach.

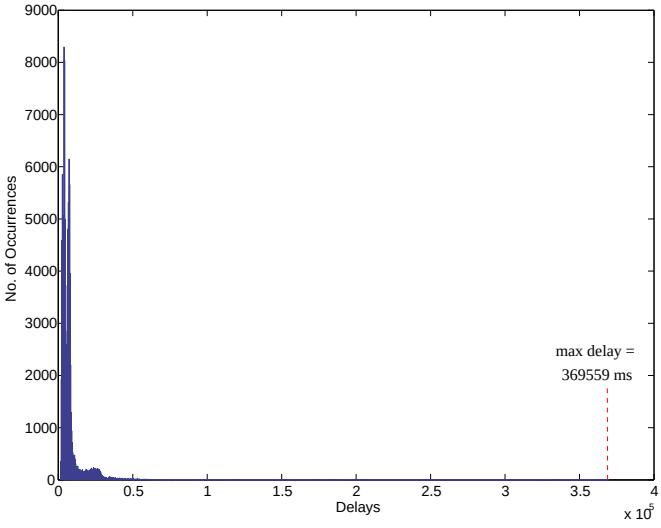

Fig. 7. Empirical distribution of end-to-end orchestration delays for 100000 simulations in the bootstrapping case.

2) T Location-Scale Sampling based Simulations: In this mode of simulation, T location-scale distributions are sampled to generate delay values for site calls. The delay values of the six Web Services were fitted with a T Location-scale distribution, giving the estimated μ, σ and ν parameters of the distribution. The pdf for this distribution is:

$$p(x) = \frac{\Gamma(\frac{\nu+1}{2})}{\sigma\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})}\left[\frac{\nu + \left(\frac{x-\mu}{2}\right)^2}{\nu}\right]^{-\left(\frac{\nu+1}{2}\right)}$$

The association of sites of CarOnLine and the Web services remains unchanged, as given in table II.

Results using hard contracts: On setting the delay contracts of each of the sites to their 99.2% quantile values, we get the end-to-end orchestration delay bound to be $K = 1469,539$ ms, which can be guaranteed for 94.53% of the cases.

Results using probabilistic soft contracts: As before, we assume zero delay for all the internal orchestration actions and perform 100,000 runs of the orchestration in this configuration. End to end orchestration delays from the simulations were recorded. In this case, the 94.53% quantile is found to be 14,658 ms.

The results are summarized in Table III.

Mode	Soft contract 94.53% quantile	Hard contract 94.53% quantile
BootStrap	23,189	44,243
T Location Dist	14,658	1,469,539

TABLE III
NO TIMEOUT CASE: COMPARISON OF DELAY QUANTILES

The time taken for the 100,000 simulations in the bootstrap mode was 37.74 sec and in the T Location-sampling mode was 42.13 sec.

Case 2: Finite Timeouts

Using hard contracts in orchestrations having timeouts raises difficulties. As an illustration, consider again Figure 3. Let K_1 and K_2 be the two hard bounds (in ms) for response times in the contracts of sites $S1$ and $S2$, respectively. Assume that timers are used to guard the two site calls, with timeout occurring at λ ms. Then, clearly, the contract that results for this orchestration entirely depends on the relative position of λ, K_1, and K_2. If $\lambda > K_i$ for $i = 1, 2$, then a timeout is supposed to never occur (unless one of the site contracts is violated). On the other hand, if $\lambda < K_i$ for $i = 1, 2$ then, even if the sites respect their contracts, this may at times be seen by the orchestration as a timeout. Clearly, using timers in combination with hard contracts makes little sense.

In contrast, probabilistic soft contracts allow using timers with no contradiction. The reason is that Monte-Carlo simulations have no problem simulating timers and their effect on the distribution of the orchestration response time. As a consequence, we only present the results from our simulations without a comparison to the hard contract based composition.

We again perform simulations in two modes: Bootstrap and T Location-scale based simulations.

1) Bootstrap based Simulations: As before, we associated each service in the CarOnLine example with delay behaviours of one of the six web services measured. The associations are the same as before, given in table II. We now have timeouts for the calls to sites GarageA and GarageB. The 99.2% delay quantiles for these two sites are 3,304 msec and 4,183 msec respectively. We perform simulations with different timeout values: 3,000, 4,000 and 5,000 msec. The results are given in table IV.

2) T-Location Scale Sampling based Simulations: We maintain the associations of table II and perform simulations by sampling the fitted T Location-scale distributions. The results of these simulations summarized in Table IV. The average time for 100,000 simulations in the bootstrap mode was 34.29 sec and in the T Location-sampling mode was 43.75 sec.

Mode	Soft contract 94.53% quantile	Timeout Value T
BootStrap	23,040	3,000
BootStrap	22,681	4,000
BootStrap	22,834	5,000
T Location Dist	13,258	3,000
T Location Dist	13,364	4,000
T Location Dist	13,582	5,000

TABLE IV
FINITE TIMEOUT CASE: DELAY QUANTILES

V. CONCLUSION

We have studied contract composition for Web services orchestrations. Our study has revealed a number of problems when relying on conventional contracts based on hard bounds—even if the latter are used in combination with percentages of contract violation. We have advocated the use of *soft* contracts instead. And we have proposed *probabilistic* contracts as a natural way to implement soft contracts.

Probabilistic soft contracts have a number of advantages. First, they compose easily, as shown by our Monte-Carlo based dimensioning tool *TOrQuE*. Second, they provide opportunity for well sound overbooking, thus avoiding pessimistic contracts. Third, they allow handling timers as part of the orchestration, a frequent and desirable practice. We stress that our *TOrQuE* tool can indeed be used for the dimensioning of realistic orchestrations, as the cost of running Monte-Carlo simulation for design space exploration is acceptable.

Future work to consider includes probabilistic soft contract *monitoring*, i.e., the detection, by the orchestration, of the violation of a site contract. Again, our approach opens avenues for this. If p is the response time distribution promised by a site as part of its contract with the orchestration, then monitoring for contract violation can be performed as follows. Let $\widehat{p}$ be the empirical distribution of the considered site, as measured on-line by the orchestration. Then, we need to design statistical tests to decide whether or not $\widehat{p} \preceq p$ holds, where $\preceq$ is the *stochastic ordering* between distributions [1]: for two random variables X and Y, $X \preceq Y$ means that, for every x, the probability that X exceeds x is less than the probability that Y exceeds the same value. In [1], statistical procedures are provided to this end.

REFERENCES

[1] G. Anderson. Nonparametric tests of stochastic dominance in income distributions. *Econometrica*, **64**(5), 1183–1193, 1996.

[2] W.M.P. van der Aalst and K.M. van Hee. Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge, MA, USA, 2002.

[3] P. Bhoj, S. Singhal, and S. Chutani. SLA Management in Federated Environments. In M. Sloman, S. Mazumdar, and E. Lupu, editors, *Proc. of Sixth IFIP/IEEE Symposium on Integrated Network Management*, IM 99, pages 293-308.

[4] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte (Editor), I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services. [BPEL4WS.] Version 1.1. 5-May-2003. 136 pages. http://xml.coverpages.org/BPELv11-May052003Final.pdf

[5] W.M.P van der Aalst. Verification of workflow nets. *Application and Theory of Petri Nets 1997*, volume 1248 of LNCS, pages 407426.

[6] Andries van Dijk. Contracting Workflows and Protocol Patterns. Business Process Management 2003, Wil M. P. van der Aalst and Arthur H. M. ter Hofstede and Mathias Weske Eds., LNCS 2678, 152-167, 2003.

[7] F. Puhlmann, M. Weske: Using the Pi-Calculus for Formalizing Workflow Patterns. In W.M.P. van der Aalst et al. (Eds.): BPM 2005, volume 3649 of LNCS, Nancy, France, Springer-Verlag (2005) 153-168.

[8] 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.

[9] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237-267, 2003.

[10] J. Arias-Fisteus, L. Sánchez Fernández, and C. Delgado Kloos. Applying model checking to BPEL4WS business collaborations. SAC 2005: 826-830.

[11] V. Firiou, J-Y. Le Boudec, D. Towsley, and Z-L. Zhang. Theories and models for Internet Quality of Service. *Proc. of the IEEE*, **90**(9), 1565–1591, 2002.

[12] Keller A., Ludwig H., The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, Vol. 11, No 1, Plenum Publishing, pp. 57-81.

[13] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon. Web Services Choreography Description Language – WS-CDL, version 1.0. http://www.w3.org/2002/ws/chor/edcopies/cdl/cdl.html

[14] METEOR-S: Semantic Web Services and Processes http://swp.semanticweb.org/

[15] R. Aggarwal, K. Verma, J. Miller and W. Milnor. Constraint Driven Web Service Composition in METEOR-S, *Proc. of IEEE International Conference on Services Computing*, SCC 2004.

[16] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, Quality of Service for Workflows and Web Service Processes, Journal of Web Semantics 2004.

[17] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. "QoS aware Middleware for Web Service Composition" *IEEE Transactions on Software Engineering*, volume 30, issue 5, 2004.

[18] X. Nguyen, R. Kowalczyk, and M. Phan. "Modelling and Solving QoS Composition Problem Using Fuzzy DisCSP" *Intl Conference on Web Services*, ICWS 2006.

[19] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. An Approach for QoS-aware Service Composition based on Genetic Algorithms. *Genetic and Evolutionary Computation Conference*, GECCO 2005.

[20] Z. Liu, M. S. Squillante, and J. L. Wolf. On Maximizing Service-Level-Agreement Profits. *ACM E-Commerce Conference*, Tampa, FL. October 2001

[21] P. Baldan, A. Corradini, U. Montanari Contextual Petri nets, asymmetric event structures and processes Information and Computation 171 (1) (2001), pp. 1-49.

[22] A.C. Davison, and D.V. Hinkley. Bootstrap Methods and their Application. Cambridge University Press, 1997.

[23] Jayadev Misra and William Cook. Computation orchestration: A Basis for Wide-Area Computing. *Journal of Software and Systems Modeling*, May 2006.

[24] W. R. Cook and J. Misra. Implementation Outline of Orc. See www.cs.utexas.edu/users/wcook/projects/orc/

[25] D. Kitchin, W. Cook, J. Misra, A. Benveniste, S. Haar, C. Jard and S. Rosario. Event Structure Semantics for Orc. *in preparation*, 2007.

[26] W. Reisig: Modeling- and Analysis Techniques for Web Services and Business Processes. FMOODS 2005: 243-258.

[27] A. Sahai, V. Machiraju, M. Sayal, Aad P. A. van Moorsel, and F. Casati. Automated SLA Monitoring for Web Services. DSOM 2002: 28-41.

[28] H. G. Song and K. Lee: sPAC (Web Services Performance Analysis Center): Performance Analysis and Estimation Tool of Web Services. Business Process Management 2005: 109-119.

[29] XMethods. http://www.xmethods.net.

[30] The *SWAN* project. `http://swan.elibel.tm.fr`