Performance-Oriented Comparison of Web Services Via Client-Specific Testing Preorders

Marco Bernardo and Luca Padovani

Università di Urbino – Italy
Istituto di Scienze e Tecnologie dell'Informazione

Abstract. The behavior of a Web service can be described by means of a contract, which is a specification of the legal interactions with the service. Given a repository of Web services, from the client viewpoint a proper service selection should be based on functional as well as non-functional aspects of the interactions. In this paper we provide a technique that enables a client both to discover compatible services and to compare them on the basis of specific performance requirements. Our technique, which is illustrated on a simple probabilistic calculus, relies on two families of client-specific probabilistic testing preorders. These are shown to be precongruences with respect to the operators of the language and not to collapse into equivalences unlike some more general probabilistic testing preorders appeared in the literature.

1 Introduction

The recent trend in Web services is fostering a computing scenario where clients must be able to search at run time services that provide specific capabilities. This scenario requires Web services to publish their capabilities in some known registry and it entails the availability of powerful search operations for capabilities. Possible capabilities that one would like to search concern the format of the exchanged messages, the protocol – or *contract* – required to interact successfully with the service, and, when considering QoS-aware Web services, capabilities describing non-functional aspects of the service.

The Web Service Description Language (WSDL) [11,10,9] and the Web Service Conversation Language (WSCL) [1] are examples of standardized technologies for describing the interface exposed by a service. Such a description includes the service location, the format (or *schema*) of the exchanged messages, the transfer mechanism to be used (e.g. SOAP-RPC, or others), and the contract. Both WSDL and WSCL documents can be published in registries [2,13] so that they can be searched and queried.

This immediately asks for a definition of *compatibility* between different published contracts. It is necessary to define precise notions of contract similarity and compatibility and use them to perform service discovery. Unfortunately, neither WSDL nor WSCL can effectively define these notions, for the very simple reason

M.M. Bonsangue and E.B. Johnsen (Eds.): FMOODS 2007, LNCS 4468, pp. 269–284, 2007.

that they do not provide any formal characterization of their contract languages. This calls for a mathematical foundation of contracts and formal relationships between clients and contracts, which have been investigated in [16,7,8].

With respect to non-functional aspects of Web services, neither WSDL nor WSCL take them into account. In fact, a few extensions have been proposed in the literature [18,15] to enrich service descriptions, in particular WSDL interfaces and UDDI registries, with QoS aspects. In some cases, a "QoS certifier" takes care of certifying the QoS claims of Web services that register themselves with it. Anyway, the QoS aspects are necessarily quantified on the basis of an "average client" interacting with the service, whereas the behavior of each specific client, especially in involved interactions, may result in significant deviations from the declared – possibly certified – quantities.

To overcome this limitation – which may cause many clients to make a wrong service selection – it is first of all necessary that the service contracts are enriched with the description of performance aspects. In fact, to make principled choices a specific client cannot only rely on claims like "the response time is 93 msec", but needs to see in the service contract more low-level performance details, like e.g. an estimate of the probability with which at a certain branching point a service behaves in a given way rather than in a different one.

In this paper we propose a technique by means of which, given a specific client and a repository of Web services whose contracts embody QoS details, the client can detect the presence of compatible services in the repository and, if any, order them on the basis of certain performance requirements that are of interest to the client.

The formal machinery that we develop to implement the technique relies on a basic weighted process calculus to describe client and service contracts. The calculus comprises weighted active and passive actions [6,3] and its only operators are termination, action prefix, and alternative composition. Weights are associated with actions to express performance aspects, with a generative interpretation in the case of active actions and a reactive interpretation in the case of passive actions [17]. As far as active actions are concerned, generative weights can be given a time-abstract interpretation (probabilities) or a continuous-time interpretation (rates of exponentially distributed durations). In the first case, the performance model underlying the interaction of a client with a service is a finite discrete-time Markov chain, while in the second case it is a finite continuous-time Markov chain [19].

A probabilistic variant of testing preorder [14] is then employed both to verify the compatibility of a service with a client and to order compatible services on the basis of client-specific performance properties. Testing preorder is an effective means to achieve the second objective in practice. In fact, a client – suitably enriched with success decorations in the appropriate places – can be viewed as a test that different services pass with different probabilities. Those probabilities precisely characterize the client-specific quality guarantees provided by the various services.

On the theoretical side, the peculiarity of our probabilistic testing preorder, which is shown to be a precongruence with respect to the operators of the basic weighted process calculus, is that of being test specific. In other words, its definition does not exhibit any universal quantification over tests. Therefore we have to do with as many test-specific probabilistic testing preorders as there are tests. An important consequence is that our probabilistic testing preorders do not collapse into equivalences. This happened for instance with the testing preorder for fully generative probabilistic processes of [12], as two processes can be in a certain relation with respect to a test and in the opposite relation with respect to another test. As another example, the testing preorder for continuous-time Markovian processes of [4] suffered from a similar problem, as it is not possible to define it in a way that is consistent with all the reward-based performance measures.

This paper is organized as follows. In Sect. 2 we define the basic weighted process calculus for describing the functional and performance aspects of client and service contracts. In Sect. 3 we define two families of client-specific probabilistic testing preorders, one for the time-abstract case and one for the continuous-time case, and we investigate their precongruence properties. In Sect. 4 we show how to use the two families of client-specific probabilistic testing preorders for compatibility verification. In Sect. 5 we exhibit some examples in which the two families of client-specific probabilistic testing preorders are used to order different services that are compatible with the same client. Finally, in Sect. 6 we provide some concluding remarks.

2 Basic Language for QoS-Aware Contracts

In this section we introduce the syntax and the semantics for a very simple weighted process calculus called WPC, which we shall use to formalize the behavior of client and service contracts in a way that takes performance aspects into account. WPC builds on a set *Name* of action names including τ for invisible actions, which will be ranged over by a, b. Its set of operators is formed only by termination, action prefix, and alternative composition.

Similarly to [6,3], an action of WPC can be either active or passive. An active action represents an activity undertaken by a process, either locally or in cooperation with other processes. By contrast, a passive action models a situation in which a process waits for another process to initiate some activity in which the former is involved as well.

Performance aspects are described by associating a positive real number – which we call weight – with each action and by assuming that the execution probability of each action is proportional to the number associated with it. More precisely, according to the terminology of [17], the choice among active actions is assumed to be generative, i.e. weights are considered across active actions with arbitrary names. By contrast, the choice among passive actions is assumed to

be reactive, i.e. weights are considered only within sets of passive actions having the same name. Thus, the choice between two passive actions having different names is nondeterministic.

An active action will be denoted by $<a, w>$ with the generative weight $w \in \mathbf{R}_{>0}$, while a passive action will be denoted by $<a, *_u>$ with the reactive weight $u \in \mathbf{R}_{>0}$. Since in WPC an invisible action can only represent a local activity, it cannot be passive, i.e. it will be of the form $<\tau, w>$. We note that the choice between two observable actions is external – in the sense that it can be influenced by the environment – independently from the fact that the actions are active or passive. Instead, the choice between two invisible actions is internal.

The generative weights associated with the active actions can be given a time-abstract interpretation or a continuous-time interpretation. In the first case, they represent non-normalized probability values and the preselection policy applies. This simply means that the choice among several simultaneously enabled active actions is solved probabilistically on the basis of action weights. In the second case, the weights represent the rates of the exponential distributions quantifying the durations of the actions. In this case the race policy applies, which means that the fastest action among the enabled ones will be executed. It can be shown that also in the second case each enabled action has an execution probability proportional to its weight. Moreover, the average sojourn time for a term turns out to be the inverse of the sum of the weights of the actions enabled by the term.

Definition 1. *The set $\mathcal{P}$ of the process terms of WPC is generated by the following syntax:*

$$\boxed{P ::= \underline{0} \mid <a, w>.P \mid <b, *_u>.P \mid P + P}$$

where $b \neq \tau$. ■

The semantics for WPC can be defined in the usual operational style, provided that the multiplicity of each transition – corresponding to the number of different proofs for the derivation of the transition – is taken into account. The reason is that the idempotency law $P + P = P$ no longer holds when dealing with probabilistic processes. As an example, in the continuous-time case, a term like $<a, 4.6>.P + <a, 4.6>.P$ is not equivalent to $<a, 4.6>.P$ but to $<a, 9.2>.P$, because the average sojourn time for $<a, 4.6>.P + <a, 4.6>.P$ is $1/9.2$.

As a consequence, the behavior of each WPC term is given by a multitransition system, whose states correspond to process terms and whose transitions are labeled with actions. Observed that the null term $\underline{0}$ cannot execute any action – hence the corresponding labeled multitransition system is just a state with no transitions – we now provide the semantic rules for the other operators of WPC:

– Action prefix: $<a, w>.P$ (resp. $<b, *_u>.P$) can execute an action named a (resp. $b \neq \tau$) and then behaves as P:

$$\boxed{<a, w>.P \xrightarrow{a,w} P \qquad <b, *_u>.P \xrightarrow{b,*_u} P}$$

– Alternative composition: $P_1 + P_2$ behaves as either P_1 or P_2 depending on whether P_1 or P_2 executes an action first:

$$\frac{P_1 \xrightarrow{a,w} P'}{P_1 + P_2 \xrightarrow{a,w} P'} \qquad \frac{P_1 \xrightarrow{b,*_u} P'}{P_1 + P_2 \xrightarrow{b,*_u} P'}$$

$$\frac{P_2 \xrightarrow{a,w} P'}{P_1 + P_2 \xrightarrow{a,w} P'} \qquad \frac{P_2 \xrightarrow{b,*_u} P'}{P_1 + P_2 \xrightarrow{b,*_u} P'}$$

Example 1. Consider a service computing the greatest common divisor and a service computing the square root. Their contracts are described in WPC as follows:

$$S_1(w_1) = <\texttt{gcd}, *_1>.<\texttt{op1}, *_1>.<\texttt{op2}, *_1>.<\texttt{res}, w_1>.<\texttt{end}, 1>.\underline{0}$$
$$S_2(w_2) = <\texttt{sqrt}, *_1>.<\texttt{op}, *_1>.(<\tau, 1>.<\texttt{res}, w_2>.<\texttt{end}, 1>.\underline{0} + <\tau, 1>.<\texttt{error}, 1>.\underline{0})$$

where we use passive actions to model messages that are sent from the client to the service, and we use active actions to model messages that are sent from the service back to the client.

The contract $S_1(w_1)$ describes the behavior of a service that computes the greatest common divisor of two positive integer numbers, with w_1 representing the performance of the service in completing the operation. The service is linear: the conversation is wrapped between actions `gcd` and `end` that delimit the actual exchange of information between client and service.

The need for an explicit `end` action to signal a terminated interaction is not immediately evident. The problem arises when a contract has the form:

$$<\tau, w'>.\underline{0} + <\tau, w''>.<a, w>.P$$

because a client interacting with a service that exposes this contract cannot distinguish a completed interaction where the service has internally decided to behave like $\underline{0}$ from an interaction where the service has internally decided to perform the a action, but it is taking a long time to respond. By providing an explicit `end` action signaling a completed interaction, the service tells the client not to wait for further messages. This way of modeling a completed interaction is consistent with the WSCL language, which accounts for an explicit termination message called "empty".

The contract $S_2(w_2)$ describes the behavior of a service that computes the square root of a real number, with w_2 representing again the performance of the service in completing the operation. After the number has been sent from the client, the service internally decides whether the operation can be completed successfully, by sending the result back to the client, or if the computation terminates either because the input is invalid (the number is less than zero) or for any other reason (the computational capacity of the service has been exceeded). Invisible actions allow us to model such kind of so-called internal choices.

Finally, we can combine the two contracts and define:

$$S_1(w_1) + S_2(w_2)$$

that describes the behavior of services providing both operations. Because of the actions `gcd` and `sqrt` that uniquely determine the kind of operation to carry on, clients can decide which operation to invoke. In other words, this is a so-called external choice. ■

3 Client-Specific Probabilistic Testing Preorders

In this section we define two families of client-specific probabilistic testing preorders for WPC and we investigate their precongruence property.

3.1 Interaction System of a Service and a Client

Given a service S and a client C both formalized in WPC, their interaction can be described by means of their parallel composition, which we denote by $S \parallel C$. If we view C as a test and we mark some of its terminal states as successful, then we can talk about the probability with which S passes the test, which corresponds to the QoS guarantee provided by S when interacting with C.

From now on, clients will thus be formalized through the set $\mathcal{P}_{\mathrm{s}}$ of terms generated by the following syntax:

$$\boxed{P ::= \underline{0} \mid \mathrm{s} \mid \sum_{i \in I} <a_i, \tilde{w}_i>.P_i}$$

where the zeroary operator "s" stands for successful termination, I is a finite non-empty set, and $\tilde{w}_i$ stands for a generative or reactive weight (in the second case $a_i \neq \tau$). The use of a guarded alternative composition operator – instead of an action prefix operator and a binary alternative composition operator – is necessary to avoid terms like $\underline{0} + \mathrm{s}$ that are ambiguous for the computation of the probability of passing a test.

The intended meaning of $S \parallel C$ is that S and C have to communicate on any observable action name. If at a certain point the set of observable action names enabled by the current derivative of S is disjoint from the set of observable action names enabled by the current derivative of C, and neither the S derivative nor the C derivative can evolve autonomously by performing an invisible action, then the service requested by C cannot be completed by S.

More precisely, in order for them to be executable, the observable active actions of the current derivative of S (resp. C) must be matched by passive actions of the current derivative of C (resp. S) having the same name. This leads to the generative-reactive synchronization mode described in [6] for time-abstract probabilistic processes and in [3] for continuous-time probabilistic processes. This synchronization mode is defined by the following two operational rules:

$$\boxed{\frac{S \xrightarrow{b,w} S' \qquad C \xrightarrow{b,*_u} C'}{S \parallel C \xrightarrow{b,w \cdot \frac{u}{weight_{\mathrm{p}}(C,b)}} S' \parallel C'} \qquad \frac{S \xrightarrow{b,*_u} S' \qquad C \xrightarrow{b,w} C'}{S \parallel C \xrightarrow{b,w \cdot \frac{u}{weight_{\mathrm{p}}(S,b)}} S' \parallel C'}}$$

where the weight of $P \in \{S, C\}$ with respect to passive actions of name $b \neq \tau$ is defined as follows:

$$\boxed{weight_{\mathrm{p}}(P, b) = \sum \{\!|\, u \mid \exists P'.\, P \xrightarrow{b,*_u} P' \,|\!\}}$$

In addition, we have two operational rules for the autonomous evolution of S (under the constraint that C has not terminated yet) and of C when performing an invisible action:

$$\boxed{\frac{S \xrightarrow{\tau,w} S' \qquad C \notin \{\underline{0}, \mathrm{s}\}}{S \parallel C \xrightarrow{\tau,w} S' \parallel C} \qquad \frac{C \xrightarrow{\tau,w} C'}{S \parallel C \xrightarrow{\tau,w} S \parallel C'}}$$

The constraint on the autonomous evolution of S is motivated by the fact that nothing can change from the point of view of passing a test once the test has reached its termination.

Definition 2. *Let $S \in \mathcal{P}$ and $C \in \mathcal{P}_{\mathrm{s}}$. The interaction system of service S and client C is process term $S \parallel C$, where we say that:*

- *A* configuration *is a state of the labeled multitransition system underlying $S \parallel C$, which is formed by a service part and a client part.*
- *A configuration is* successful *iff its client part is "*s*".* ■

3.2 Computations: Execution Probability and Average Duration

A computation is a sequence of transitions that can be executed starting from $S \parallel C$. We say that two computations are independent of each other if it is not the case that one of them is a proper prefix of the other one. Moreover we say that a computation is successful if so is its last configuration. We denote by $\mathcal{C}(S, C)$, $\mathcal{IC}(S, C)$, and $\mathcal{SC}(S, C)$ the multisets of the computations, of the independent computations, and of the successful computations of $S \parallel C$, respectively.[1]

Let us define the length of a computation as the number of transitions occurring in it. From the fact that recursion is not allowed and the finitely-branching structure of S and C, it immediately follows that $\mathcal{C}(S, C)$ is finite and all of its computations have finite length. Moreover, $\mathcal{SC}(S, C) \subseteq \mathcal{IC}(S, C)$ because of the maximality of the length of the successful computations.

Two important quantities that can be associated with each computation are its execution probability and – in the continuous-time case – its average duration. Below we provide their inductive definitions.

Definition 3. *Let $S \in \mathcal{P}$, $C \in \mathcal{P}_{\mathrm{s}}$, and $c \in \mathcal{C}(S, C)$. The probability of executing c is the product of the execution probabilities of the transitions of c, which is defined by induction on the length of c through the following $\mathbf{R}_{]0,1]}$-valued function:*

$$\boxed{prob(c) = \begin{cases} 1 & \text{if } length(c) = 0 \\ \dfrac{w}{weight_{\mathrm{t}}(S \parallel C)} \cdot prob(c') & \text{if } c \equiv S \parallel C \xrightarrow{a,w} c' \end{cases}}$$

[1] Since transitions have multiplicities, computations also have multiplicities.

where the total weight of $S \parallel C$ is defined as follows:

$$\boxed{weight_{\mathrm{t}}(S \parallel C) = \sum\{\!|\, w \mid \exists a, S', C'.\, S \parallel C \xrightarrow{a,w} S' \parallel C' \,|\!\}}$$

We also define the probability of executing a computation of K as:

$$\boxed{prob(K) = \sum_{c \in K} prob(c)}$$

for all $K \subseteq \mathcal{IC}(S, C)$. ■

Definition 4. *Let $S \in \mathcal{P}$, $C \in \mathcal{P}_{\mathrm{s}}$, and $c \in \mathcal{C}(S, C)$. Assume a continuous-time interpretation for all the generative weights occurring in S and C. The average duration of c is the sequence of the average sojourn times[2] in the states traversed by c, which is defined by induction on the length of c through the following $(\mathbf{R}_{>0})^*$-valued function:*

$$\boxed{time(c) = \begin{cases} \varepsilon & \text{if } length(c) = 0 \\ \frac{1}{weight_{\mathrm{t}}(S \parallel C)} \circ time(c') & \text{if } c \equiv S \parallel C \xrightarrow{a,w} c' \end{cases}}$$

where ε is the empty average duration and $\circ$ is the sequence concatenation operator. We also define the multiset of the computations of K whose average duration is not greater than θ as:

$$\boxed{K_{\leq \theta} = \{\!|\, c \in K \mid length(c) \leq length(\theta) \wedge \forall i = 1, \ldots, length(c).\, time(c)[i] \leq \theta[i] \,|\!\}}$$

for all $K \subseteq \mathcal{C}(S, C)$ and $\theta \in (\mathbf{R}_{>0})^*$. ■

Example 2. Consider three potential clients of the services $S_1(w_1)$ and $S_2(w_2)$ introduced in Ex. 1, whose contracts are described in WPC as follows:

$$C_{1,\mathrm{s}} = \texttt{<gcd},1\texttt{>.<op1},1\texttt{>.<op2},1\texttt{>.<res},*_1\texttt{>.<end},*_1\texttt{>.s}$$
$$C_{2,\mathrm{s}} = \texttt{<sqrt},1\texttt{>.<op},1\texttt{>.(<res},*_1\texttt{>.<end},*_1\texttt{>.s} + \texttt{<error},*_1\texttt{>.s)}$$
$$C_{3,\mathrm{s}} = \texttt{<sqrt},1\texttt{>.<op},1\texttt{>.<res},*_1\texttt{>.<end},*_1\texttt{>.s}$$

It is easy to see that:

$$\mathcal{SC}(S_1(w_1), C_{1,\mathrm{s}}) = \{S_1(w_1) \parallel C_{1,\mathrm{s}} \xrightarrow{\texttt{gcd},1} \cdot \xrightarrow{\texttt{op1},1} \cdot \xrightarrow{\texttt{op2},1} \cdot \xrightarrow{\texttt{res},w_1} \cdot \xrightarrow{\texttt{end},1} \underline{0} \parallel \mathrm{s}\}$$

$$\mathcal{SC}(S_2(w_2), C_{2,\mathrm{s}}) = \{S_2(w_2) \parallel C_{2,\mathrm{s}} \xrightarrow{\texttt{sqrt},1} \cdot \xrightarrow{\texttt{op},1} \cdot \xrightarrow{\tau,1} \cdot \xrightarrow{\texttt{res},w_2} \cdot \xrightarrow{\texttt{end},1} \underline{0} \parallel \mathrm{s},$$
$$S_2(w_2) \parallel C_{2,\mathrm{s}} \xrightarrow{\texttt{sqrt},1} \cdot \xrightarrow{\texttt{op},1} \cdot \xrightarrow{\tau,1} \cdot \xrightarrow{\texttt{error},1} \underline{0} \parallel \mathrm{s}\}$$

$$\mathcal{SC}(S_2(w_2), C_{3,\mathrm{s}}) = \{S_2(w_2) \parallel C_{3,\mathrm{s}} \xrightarrow{\texttt{sqrt},1} \cdot \xrightarrow{\texttt{op},1} \cdot \xrightarrow{\tau,1} \cdot \xrightarrow{\texttt{res},w_2} \cdot \xrightarrow{\texttt{end},1} \underline{0} \parallel \mathrm{s}\}$$

from which we derive that $prob(\mathcal{SC}(S_1(w_1), C_{1,\mathrm{s}}) = prob(\mathcal{SC}(S_2(w_2), C_{2,\mathrm{s}}) = 1$ and $prob(\mathcal{SC}(S_2(w_2), C_{3,\mathrm{s}})) = \frac{1}{2}$. ■

[2] The average sojourn time of a term is the inverse of the sum of the weights of the actions enabled by the term.

3.3 Preorder Definition

We are now in a position to define two families of client-specific probabilistic testing preorders – one for the time-abstract case and one for the continuous-time case – which can be used by the clients to order the services on the basis of the QoS levels resulting from the interaction with them. This is helpful from the client viewpoint to select the service providing the best performance guarantees.

Definition 5. *Let $S_1, S_2 \in \mathcal{P}$ and $C \in \mathcal{P}_{\rm s}$. We say that S_1 is probabilistic testing less than S_2 with respect to C in the time-abstract case, written $S_1 \sqsubseteq^{C}_{\rm PT,ta} S_2$, iff:*

$$prob(\mathcal{SC}(S_1, C)) \leq prob(\mathcal{SC}(S_2, C))$$

■

Definition 6. *Let $S_1, S_2 \in \mathcal{P}$ and $C \in \mathcal{P}_{\rm s}$. Assume a continuous-time interpretation for all the generative weights occurring in S_1, S_2, and C. We say that S_1 is probabilistic testing less than S_2 with respect to C in the continuous-time case, written $S_1 \sqsubseteq^{C}_{\rm PT,ct} S_2$, iff for all $\theta \in (\mathbb{R}_{>0})^*$:*

$$prob(\mathcal{SC}_{\leq\theta}(S_1, C)) \leq prob(\mathcal{SC}_{\leq\theta}(S_2, C))$$

■

3.4 Precongruence Property

We conclude by proving that the two families of client-specific probabilistic testing preorders are precongruences with respect to the operators of WPC. The result will be presented for the time-abstract case only, as in the continuous-time case it is similar.

As far as action prefix is concerned, the result is formulated in a non-standard way. The reason is that we do not have to do with a standard testing preorder with universal quantification over all tests, but with a family of client-specific testing preorders. Thus, whenever two interaction systems $S_1 \parallel C$ and $S_2 \parallel C$ perform a transition that causes C to evolve to C', the two derivative interaction systems can no longer be compared with respect to C, but have to be compared with respect to C'.

In the following, we denote by $C \stackrel{p}{\Longrightarrow} C'$ the fact that client C can evolve to C' with probability p after executing a finite sequence of zero or more invisible transitions. The probability p is computed as a product of ratios, each of which relates to an invisible transition in the sequence and is given by the weight of the transition itself divided by the sum of the weights of all the invisible transitions departing from the source state of the considered transition. In the case in which $C' = C$, we let $p = 1$.

Theorem 1. *Let $S_1, S_2 \in \mathcal{P}$ and $C \in \mathcal{P}_{\rm s}$. Whenever for all $C \stackrel{p_i}{\Longrightarrow} C_i \stackrel{b,*_{u_{i,j}}}{\longrightarrow} C_{i,j}$ it holds $S_1 \sqsubseteq^{C_{i,j}}_{\rm PT,ta} S_2$, then $<b,w>.S_1 \sqsubseteq^{C}_{\rm PT,ta} <b,w>.S_2$ for all $b \neq \tau$ and $w \in \mathbb{R}_{>0}$.* ■

Theorem 2. *Let $S_1, S_2 \in \mathcal{P}$ and $C \in \mathcal{P}_{\rm s}$. Whenever for all $C \stackrel{p_i}{\Longrightarrow} C_i \stackrel{b,w_{i,j}}{\longrightarrow} C_{i,j}$ it holds $S_1 \sqsubseteq^{C_{i,j}}_{\rm PT,ta} S_2$, then $<b,*_u>.S_1 \sqsubseteq^{C}_{\rm PT,ta} <b,*_u>.S_2$ for all $b \neq \tau$ and $u \in \mathbb{R}_{>0}$.* ■

Theorem 3. *Let $S_1, S_2 \in \mathcal{P}$ and $C \in \mathcal{P}_{\rm s}$. Whenever for all $C \stackrel{p_i}{\Longrightarrow} C_i$ it holds $S_1 \sqsubseteq^{C_i}_{\rm PT,ta} S_2$, then $<\tau,w>.S_1 \sqsubseteq^{C}_{\rm PT,ta} <\tau,w>.S_2$ for all $w \in \mathbb{R}_{>0}$.* ■

In the case of the alternative composition, precongruence is achieved only for pairs of interaction systems satisfying certain weight-related constraints. More precisely, such constraints are concerned with the total active weight of a service S when evolving locally or interacting with a client C:

$$W_{\rm s}(S,C) = \sum_{S \stackrel{a,w}{\longrightarrow} S'} \{\!|\, w \mid a = \tau \vee \exists u, C'.\, C \stackrel{a,*_u}{\longrightarrow} C' \,|\!\}$$

and with the total active weight of a client C when interacting with a service S alternative to another service R:

$$W_{\rm c}(C,S,R) = \sum_{C \stackrel{b,w}{\longrightarrow} C'} \{\!|\, w \cdot \frac{weight_{\rm p}(S,b)}{weight_{\rm p}(S,b)+weight_{\rm p}(R,b)} \mid \exists u, S'.\, S \stackrel{b,*_u}{\longrightarrow} S' \,|\!\}$$

Theorem 4. *Let $S_1, S_2 \in \mathcal{P}$ and $C \in \mathcal{P}_{\rm s}$. Whenever for all $C \stackrel{p_i}{\Longrightarrow} C_i$ it holds $S_1 \sqsubseteq^{C_i}_{\rm PT,ta} S_2$ with $W_{\rm s}(S_1,C_i) = W_{\rm s}(S_2,C_i)$ and $weight_{\rm p}(S_1,b) = weight_{\rm p}(S_2,b)$ for all $b \neq \tau$ such that C_i enables an active b-action, then $S_1 + S \sqsubseteq^{C}_{\rm PT,ta} S_2 + S$ and $S + S_1 \sqsubseteq^{C}_{\rm PT,ta} S + S_2$ for all $S \in \mathcal{P}$.* ■

The constraint "$W_{\rm s}(S_1,C_i) = W_{\rm s}(S_2,C_i)$" is strictly necessary to achieve precongruence with respect to alternative composition. Consider e.g. the following terms:

$$\begin{aligned} S_1 &= <a,40>.\underline{0} + <b,60>.\underline{0} \\ S_2 &= <a,5>.\underline{0} + <b,5>.\underline{0} \\ S &= <a,1>.\underline{0} + <b,9>.\underline{0} \\ C &= <a,*_1>.{\rm s} + <b,*_1>.\underline{0} \end{aligned}$$

where the only invisible transition sequence of the client is $C \stackrel{1}{\Longrightarrow} C$ with:

$$W_{\rm s}(S_1,C) = 40 + 60 = 100 \neq 10 = 5 + 5 = W_{\rm s}(S_2,C)$$

Then we have:

$$prob(\mathcal{SC}(S_1,C)) = \tfrac{40}{100} = 0.4 < 0.5 = \tfrac{5}{10} = prob(\mathcal{SC}(S_2,C))$$

but:

$$prob(\mathcal{SC}(S_1+S,C)) = \tfrac{40}{110} + \tfrac{1}{110} \approx 0.37 > 0.3 = \tfrac{5}{20} + \tfrac{1}{20} = prob(\mathcal{SC}(S_2+S,C))$$

Similarly, the constraint "$weight_{\rm p}(S_1,b) = weight_{\rm p}(S_2,b)$ for all $b \neq \tau$ such that C_i enables an active b-action" is strictly necessary. Consider e.g. the following terms:

$$\begin{aligned} S_1 &= <a,4>.\underline{0} + <b,6>.\underline{0} + <c,*_1>.\underline{0} \\ S_2 &= <a,5>.\underline{0} + <b,5>.\underline{0} + <c,*_{50}>.\underline{0} \\ S &= <c,*_{55}>.<d,*_1>.\underline{0} \\ C &= <a,*_1>.{\rm s} + <b,*_1>.\underline{0} + <c,10>.<d,10>.{\rm s} \end{aligned}$$

where the only invisible transition sequence of the client is $C \stackrel{1}{\Longrightarrow} C$ with:

$$W_{\mathrm{s}}(S_1, C) = 4 + 6 = 10 = 5 + 5 = W_{\mathrm{s}}(S_2, C)$$

and:

$$weight_{\mathrm{p}}(S_1, c) = 1 \neq 50 = weight_{\mathrm{p}}(S_2, c)$$

Then we have:

$$prob(\mathcal{SC}(S_1, C)) = \tfrac{4}{20} = 0.2 < 0.25 = \tfrac{5}{20} = prob(\mathcal{SC}(S_2, C))$$

but:

$$prob(\mathcal{SC}(S_1 + S, C)) = \tfrac{4}{20} + \tfrac{10}{20} \cdot \tfrac{55}{56} \approx 0.69 > \\ > 0.51 \approx \tfrac{5}{20} + \tfrac{10}{20} \cdot \tfrac{55}{105} = prob(\mathcal{SC}(S_2 + S, C))$$

4 Compatibility Verification

The selection of the service providing the best performance guarantees for a client has to be preceded by a phase during which the client searches the Web service registry for all the services that are compatible with it. Such services are the ones that ensure the complete satisfaction of the client request.

Compatibility is a functional property that can be verified with the time-abstract family of client-specific probabilistic testing preorders. The first step consists of building a canonical service that ensures the termination of the client along each of its branches. The second step consists of searching the Web service registry for all the services that – with respect to a variant of the client in which all of its terminal states are made successful – are not less than the canonical service. In other words, the canonical service is the search key for the Web service registry of the considered client.

The canonical service is formalized as the dual of the client, which is obtained from the client by making passive (resp. active) all of its observable active (resp. passive) actions, by eliminating all of its invisible actions, and by changing to $\underline{0}$ all of its successful terminal states. All the generative and reactive weights occurring in the dual are set to 1, as their values are unimportant for the sake of termination. In the following we denote by $obs(C)$ the fact that at least one observable action occurs inside client C.

Definition 7. *Let $S \in \mathcal{P}$, $C \in \mathcal{P}_{\mathrm{s}}$, and C_{s} be the everywhere-successful variant of C. We say that S is* compatible *with C iff:*

$$prob(\mathcal{SC}(S, C_{\mathrm{s}})) = 1$$ ■

Definition 8. *Let $C \in \mathcal{P}_{\mathrm{s}}$. The* dual *of C is defined by induction on the syntactical structure of C as follows:*

$$dual(\underline{0}) = \underline{0}$$
$$dual(\mathrm{s}) = \underline{0}$$
$$dual(\sum_{i \in I} <b_i, *_{u_i}>.C_i + \sum_{j \in J} <b_j, w_j>.C_j + \sum_{k \in K} <\tau, w_k>.C_k) = \\ = \sum_{i \in I} <b_i, 1>.dual(C_i) + \sum_{j \in J} <b_j, *_1>.dual(C_j) + \sum_{k \in K, obs(C_k)} dual(C_k)$$

where: $b_j \neq \tau$ for $j \in J$; I, J, and K are pairwise disjoint with $I \cup J \cup K$ finite and non-empty; the term on the right-hand side of the last clause is $\underline{0}$ if all the three index sets are empty. ■

Lemma 1. *Let $C \in \mathcal{P}_{\mathrm{s}}$. Whenever $dual(C)$ is deterministic, then $dual(C) \parallel C_{\mathrm{s}}$ has as many maximal computations as C_{s} and all of them are successful.* ■

Note that the determinism of $dual(C)$ is essential. Consider e.g. the following pair composed of a client and its dual:

$$C = <\tau,1>.<a,1>.<b,1>.\underline{0} + <\tau,1>.<a,1>.<c,1>.\underline{0}$$
$$dual(C) = <a,*_1>.<b,*_1>.\underline{0} + <a,*_1>.<c,*_1>.\underline{0}$$

Then $dual(C) \parallel C_{\mathrm{s}}$ deadlocks after the interaction of the first (resp. second) passive a-action of $dual(C)$ with the second (resp. first) active a-action of C_{s}.

Theorem 5. *Let $S \in \mathcal{P}$ and $C \in \mathcal{P}_{\mathrm{s}}$. Whenever $dual(C)$ is deterministic and $dual(C) \sqsubseteq^{C_{\mathrm{s}}}_{\mathrm{PT,ta}} S$, then S is compatible with C.* ■

Example 3. Consider the three clients whose everywhere-successful variants are shown in Ex. 2. In order to check the compatibility of these clients with respect to the services defined in Ex. 1 we have to compute their dual contracts:

$$dual(C_1) = <\texttt{gcd},*_1>.<\texttt{op1},*_1>.<\texttt{op2},*_1>.<\texttt{res},1>.<\texttt{end},1>.\underline{0}$$
$$dual(C_2) = <\texttt{sqrt},*_1>.<\texttt{op},*_1>.(<\texttt{res},1>.<\texttt{end},1>.\underline{0} + <\texttt{error},1>.\underline{0})$$
$$dual(C_3) = <\texttt{sqrt},*_1>.<\texttt{op},*_1>.<\texttt{res},1>.<\texttt{end},1>.\underline{0}$$

Now we have that:

$$\mathcal{SC}(dual(C_2), C_{2,\mathrm{s}}) = \{\ dual(C_2) \parallel C_{2,\mathrm{s}} \xrightarrow{\texttt{sqrt},1} \cdot \xrightarrow{\texttt{op},1} \cdot \xrightarrow{\texttt{res},1} \cdot \xrightarrow{\texttt{end},1} \underline{0} \parallel \mathrm{s},$$
$$dual(C_2) \parallel C_{2,\mathrm{s}} \xrightarrow{\texttt{sqrt},1} \cdot \xrightarrow{\texttt{op},1} \cdot \xrightarrow{\texttt{error},1} \underline{0} \parallel \mathrm{s}\}$$

hence:

$$1 = prob(\mathcal{SC}(dual(C_2), C_{2,\mathrm{s}})) \leq prob(\mathcal{SC}(S_2(w_2), C_{2,\mathrm{s}}) = 1$$

that is:

$$dual(C_2) \sqsubseteq^{C_{2,\mathrm{s}}}_{\mathrm{PT,ta}} S_2(w_2)$$

from which we conclude that service $S_2(w_2)$ is compatible with client C_2. On the other hand:

$$\mathcal{SC}(dual(C_3), C_{3,\mathrm{s}}) = \{dual(C_3) \parallel C_{3,\mathrm{s}} \xrightarrow{\texttt{sqrt},1} \cdot \xrightarrow{\texttt{op},1} \cdot \xrightarrow{\texttt{res},1} \cdot \xrightarrow{\texttt{end},1} \underline{0} \parallel \mathrm{s}\}$$

and

$$\tfrac{1}{2} = prob(\mathcal{SC}(S_2(w_2), C_{3,\mathrm{s}})) < prob(\mathcal{SC}(dual(C_3), C_{3,\mathrm{s}})) = 1$$

that is service $S_2(w_2)$ is *not* compatible with client C_3. Indeed, the client C_3 blindly assumes that the service always completes the operation successfully, but this assumption may prove fatal if the service proposes an `error` action. By similar arguments, it is easy to verify that $S_1(w_1)$ is compatible with C_1, and that $S_1(w_1) + S_2(w_2)$ is compatible with both C_1 and C_2. ■

5 Selecting the Best Compatible Service

While the time-abstract family of client-specific probabilistic testing preorders allows us to reason about the probability that a client interacts with a service following a given computation, the family of continuous-time preorders allows us to reason about the average duration of any of such computations. In this section we present a few examples that show how the continuous-time preorders can be used to sort compatible services according to their performance, and we stress the importance of the client's contract in the selection of the best service.

Example 4. We have seen that $S_1(w_1) + S_2(w_2)$ is compatible with C_2. Consider now the following variant of C_2:

$$C_2' = <\texttt{sqrt}, 1>.<\texttt{op}, 1>.(<\texttt{res}, *_1>.<\texttt{end}, *_1>.\mathrm{s} + <\texttt{error}, *_1>.\underline{0})$$

Then, for all $c \in \mathcal{SC}(S_1(w_1) + S_2(w_2), C_2')$, we have:

$$time(c) = 1 \circ 1 \circ \tfrac{1}{2} \circ \tfrac{1}{w_2} \circ 1$$

from which we notice that greater values for the parameter w_2 guarantee smaller interaction times with the service. ■

Example 5. Observed that for all $c \in \mathcal{SC}(S_1(w_1) + S_2(w_2), C_{1,\mathrm{s}})$ we have:

$$time(c) = 1 \circ 1 \circ 1 \circ \tfrac{1}{w_1} \circ 1$$

it is easy to find w_1', w_2', w_1'', w_2'' such that:

$$S_1(w_1') + S_2(w_2') \sqsubseteq_{\mathrm{PT,ct}}^{C_2'} S_1(w_1'') + S_2(w_2'')$$

and:

$$S_1(w_1'') + S_2(w_2'') \sqsubseteq_{\mathrm{PT,ct}}^{C_{1,\mathrm{s}}} S_1(w_1') + S_2(w_2')$$

that is, the relative ordering between services may depend upon clients. This example shows that the usual probabilistic testing preorder, with universal quantification over all the tests, is not suitable to be used in our framework for selecting the best service. ■

Example 6. In previous work that relate the contracts of different services [7,8,16], services are typically ordered according to their ability of guaranteeing the termination of the client. In our framework such a relation can be roughly stated as follows (recall that C_{s} is the everywhere-successful variant of C, see Def. 7):

$$S \preceq S' \qquad \text{iff} \qquad \forall C.\, prob(\mathcal{SC}(S, C_{\mathrm{s}})) = 1 \Rightarrow prob(\mathcal{SC}(S', C_{\mathrm{s}})) = 1$$

meaning that the set of clients that S is compatible with is a subset of the set of clients that S' is compatible with. If we consider a service whose contract is:

$$S_3 = <\texttt{sqrt}, *_1>.<\texttt{real}, *_1>.<\texttt{error}, 1>.\underline{0}$$

we can state, for instance, that $S_2(w_2) \preceq S_3$. Indeed, a client that successfully terminates when interacting with $S_2(w_2)$ must take into account *all* of the

possible behaviors of $S_2(w_2)$. Since, roughly speaking, contract S_3 is "more deterministic" than $S_2(w_2)$, a client that successfully terminates when interacting with $S_2(w_2)$ does so also when interacting with S_3. However, it is hardly the case that S_3 can be considered "better" than $S_2(w_2)$, as it simply reports a failure regardless of the client's input. While $\preceq$ makes sense from a purely functional point of view (client termination is guaranteed if $S_2(w_2)$ is replaced by S_3) it makes little sense when QoS aspects are taken into account. By appropriately placing the "s" operator in the client's contract, both abstract-time and continuous-time preorder families can be used for sorting services according to the client's expectations (obtaining a result rather than an exception). ■

6 Conclusion

In this paper we have presented a technique for using a simple weighted process calculus to reason about the compatibility and the performance of services with respect to potential clients. The technique is directly related to and extends previous work on contracts [7,8] and session types [16]. On the practical side, one contribution of the paper is to provide a formal foundation that subsumes and refines existing mechanisms for specifying and assessing QoS aspects of Web services, by associating performance parameters with the single actions occurring during the conversation between a client and a service, rather than with the service as a whole. On the theoretical side, we have provided a motivation for the study of test-specific relations, which do not collapse into equivalences. This allows us to use such relations for ordering services in non-trivial ways according to a specific client's expectations, so as to maximize the client's satisfaction.

There are several directions for further investigations, we mention three of them. First, the weighted process calculus presented in this paper can be extended with a recursion operator, so as to make the language suitable for modeling more realistic scenarios where clients and services perform arbitrarily long interactions adhering to some regular pattern. It is reasonable to expect that this extension does not significantly affect the theory developed so far, and that the results proved in the finite case still hold once the usual annoyances deriving from recursion (such as divergence) have been appropriately taken care of.

Second, the notion of dual contract that we have formalized in Sect. 4 only provides a sufficient condition that guarantees the termination of a client when interacting with a service, however there is strong evidence that this notion can be relaxed. Since the dual contract is used as a search key in a Web service registry, it is desirable to find the *smallest* (or *principal*) key so as to maximize the number of services that are found to be greater than or equal to the key, according to the time-abstract preorder.

Third, we have not taken into account any aspect concerning the *composition* of Web services. Because of their very nature, it is often the case that several Web services have to be assembled together to accomplish a given task. Hence, it is interesting to investigate whether (some variant of) the weighted process calculus presented in this paper is suitable to reason about QoS aspects of

compound services. In this respect, the fact that the probabilistic testing preorders happen to be precongruences with respect to the operators of the process calculus (action prefix and alternative composition) is particularly important, as this property guarantees that the substitution of a component service with another providing better performance does not compromise the performance of the compound service as a whole.

References

1. Banerji, A., Bartolini, C., Beringer, D., Chopella, V., et al.: Web Services Conversation Language (wscl) 1.0", 2002. http://www.w3.org/TR/2002/NOTE-wscl10-20020314 (2002)
2. Beringer, D., Kuno, H., Lemon, M.: Using wscl in a uddi Registry 1.0, UDDI Working Draft Best Practices Document, 2001. http://xml.coverpages.org/HP-UDDI-wscl-5-16-01.pdf (2001)
3. Bernardo, M., Bravetti, M.: Performance Measure Sensitive Congruences for Markovian Process Algebras". Theoretical Computer Science 290, 117–160 (2003)
4. Bernardo, M., Cleaveland, R.: "A Theory of Testing for Markovian Processes. In: Palamidessi, C. (ed.) CONCUR 2000, LNCS, vol. 1877, pp. 305–319. Springer, Heidelberg (2000)
5. Booth, D., Liu, C.K.: Web Services Description Language wsdl Version 2.0 Part 0: Primer, 2006. http://www.w3.org/TR/2006/CR-wsdl20-primer-20060327 (2006)
6. Bravetti, M., Aldini, A.: Discrete Time Generative-Reactive Probabilistic Processes with Different Advancing Speeds". Theoretical Computer Science 290, 355–406 (2003)
7. Carpineti, S., Castagna, G., Laneve, C., Padovani, L.: "A Formal Account of Contracts for Web Services". In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006, LNCS, vol. 4184, pp. 148–162. Springer, Heidelberg (2006)
8. Castagna, G., Gesbert, N., Padovani, L.: A Theory of Contracts for Web Services. In: Proc. of the 5th ACM SIGPLAN Workshop on Programming Language Technologies for XML (PLAN-X 2007), pp. 37–49. ACM Press, New York (2007)
9. Chinnici, R., Haas, H., Lewis, A.A., Moreau, J.-J., et al.: Web Services Description Language (wsdl) Version 2.0 Part 2: Adjuncts, 2006. http://www.w3.org/TR/2006/CR-wsdl20-adjuncts-20060327 (2006)
10. Chinnici, R., Moreau, J.-J., Ryman, A., Weerawarana, S.: Web Services Description Language (wsdl) Version 2.0 Part 1: Core Language 2006. http://www.w3.org/TR/2006/CR-wsdl20-20060327 (2006)
11. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (wsdl) 1.1, 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315 (2001)
12. Cleaveland, R., Dayar, Z., Smolka, S.A., Yuen, S.: Testing Preorders for Probabilistic Processes. Information and Computation 154, 93–148 (1999)
13. Colgrave, J., Januszewski, K.: Using wsdl in a uddi Registry, technical note, 2004. http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm (2004)
14. de Nicola, R., Hennessy, M.: Testing Equivalences for Processes. in Theoretical Computer Science 34, 83–133 (1983)
15. Dustdar, S., Schreiner, W.: A Survey on Web Services Composition. International Journal of Web. and Grid Services 1, 1–30 (2005)

16. Gay, S., Hole, M.: Subtyping for Session Types in the π-calculus. Acta Informatica 42, 191–225 (2005)
17. Van Glabbeek, R.J., Smolka, S.A., Steffen, B.: Reactive, Generative and Stratified Models of Probabilistic Processes. Information and Computation 121, 59–80 (1995)
18. Ran, S.: A Model for Web Services Discovery with QoS. ACM SIGecom Exchanges 4, 1–10 (2003)
19. Stewart, W.J.: Introduction to the Numerical Solution of Markov Chains. Princeton University Press, Princeton (1994)