Monotonicity in Service Orchestrations*

Anne Bouillard[1], Sidney Rosario[2], Albert Benveniste[2], and Stefan Haar[3]

[1] ENS Cachan, IRISA, Université Européenne de Bretagne, Bruz France
[2] IRISA/INRIA, Campus de Beaulieu, Rennes France
[3] INRIA Saclay, ENS Cachan, France

Abstract. Web Service orchestrations are compositions of different Web Services to form a new service. The services called during the orchestration guarantee a given Quality of Service (QoS) to the orchestrator, usually in the form of contracts. These contracts can then be used by the orchestrator to deduce the contract it can offer to its own clients, by performing contract composition. An implicit monotonicity assumption in contract based QoS management is: "the better the component services perform, the better the orchestration's performance will be".
In some orchestrations, however, monotonicity can be violated, i.e., the performance of the orchestration improves when the performance of a component service degrades. This is highly undesirable since it can render the process of contract composition inconsistent.
In this paper we formally define monotonicity for orchestrations modelled by Colored Occurrence Nets (CO-nets) and we characterize the classes of monotonic orchestrations. Contracts can be formulated as hard, possibly nondeterministic, guarantees, or alternatively as probabilistic guarantees. Our work covers both cases. We show that few orchestrations are indeed monotonic, mostly because of complex interactions between control, data, and timing. We also provide user guidelines to get rid of non-monotonicity when designing orchestrations.

1 Introduction

Web Services and their compositions are being widely used to build distributed applications over the web. Web Service orchestrations are compositions of Web Services to form an aggregate, and usually more complex, Web Service. Different formalisms have been proposed for orchestrating Web Services, the most popular amongst these is the Business Process Execution Language (BPEL) [3]. Another such formalism is Orc [7], a small and elegant language equipped with extensive semantics work [6, 12]. Various other models have been used either to directly model orchestrations, or as a semantic domain for some formalisms; see for example the Petri Nets based WorkFlow Nets [13].

Though the main focus of the existing models is to capture the functional aspects of service and their compositions, the non-functional - also called Quality of Service (QoS) - aspects also need to be considered. The QoS of a service

* 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.

is characterised by different metrics - called QoS parameters - , *e.g.,* latency, availability, throughput, security, etc. QoS management is usually based on the notion of a Service Level Agreement (SLA) or *contract*, which specifies constraints on the QoS parameters of the service. A typical service contract could be : *for 95% of the requests, the response time will be less than 5ms.* The WSLA Standard [5] is one such proposition for specifying QoS through SLAs.

In service orchestrations, contracts are agreements made between the orchestrator and the different services called by the orchestrator (also called *sub-contractors*) which formalise the duties and responsibilities for each of them. The orchestrator can then compose all the contracts with its sub-contractors, to help it propose a contract to its own clients. This process is called *contract composition.* In [10] we introduced the notion of *probabilistic contracts* to formalise the QoS behaviour of services — the work of [10] focused on latency. We showed how these contracts can be composed to get the orchestration's contract. We also showed that there is room for *overbooking* the orchestrator's resources.

Contract based QoS management in orchestrations relies on the implicit assumption that if each of the sub-contractor meets its contract's objectives, then so does the orchestrator. Vice-versa, a sub-contractor breaching its contract can cause the orchestrator to breach the contract with its clients. Thus the whole philosophy behind contracts is that the better the sub-contractors behave, the better the overall orchestration will meet its contract. In fact, the authors themselves have developed their past work [10] based on this credo ... until they discovered that this implicit assumption could easily be falsified. Why so?

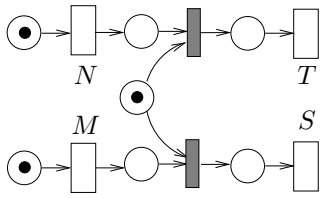

Fig. 1. A non-monotonic orchestration

As an example, consider the orchestration modeled by the Petri net in Figure 1. Services M and N are first called in parallel. If M responds first, service S is next called and the response of N is ignored. If N responds first, T is called and not S. Let δ_i denote the response time of site i. Assume the following delay behaviour: $\delta_M < \delta_N$ and $\delta_S \gg \delta_T$. Since M responds faster, the end-to-end orchestration delay is $d_0 = \delta_M + \delta_S$. Now let service M behaves slightly 'badly', *i.e* delay δ_M increases and becomes slightly greater than δ_N. Now service T is called and the new orchestration delay is $d_1 = \delta_N + \delta_T$. But since $\delta_S \gg \delta_T$, d_1 is in fact lower than d_0. This orchestration is *non-monotonic* since increasing the latency of one of its components can decrease the end-to-end latency of the orchestration. So, what is the nature of the difficulty?

"Simple" composed Web services are such that QoS aspects do not interfere with functional aspects and do not interfere with each other. Their flow of control is typically rigid and does not involve if-then-else branches. For such cases, latencies will compose gently and will not cause pathologies as shown above. However, as evidenced by the rich constructions offered by BPEL, orchestrations and choreographies can have branching based on data and QoS values, various kinds of exceptions, and timers. With such flexibility, non-monotonicity such as that exhibited by the example of Figure 1 can very easily occur.

Lack of monotonicity impairs using contracts for the compositional management of QoS. Surprisingly enough, this fact does not seem to have been noticed in the literature.

In this paper we classify orchestrations based on their monotonic characteristics. We focus on latency, although other aspects of QoS are discussed as well. Section 2 informally introduces the notion of monotonicity with examples. In Section 3 we recall the definition of Petri nets and introduce our model, Orch-Net. A formal definition of monotonicity and a characterisation of monotonic orchestrations is then given in Section 4. Section 5 extends the notion of monotonicity to nets whose transitions' delays are probability distributions. Section 6 gives a few ideas to avoid the problem of non-monotonicity and Section 7 concludes. Proofs of non-trivial results are deferred to the appendix.

2 Examples for Non-monotonic Orchestrations

In this section we look at sample orchestrations and illustrate the concept of (non) monotonicity using them.

The Travel Planner orchestration: The orchestration to the left in Figure 2 is inspired by [14]. A client calls the Travel Planner orchestration with a city he

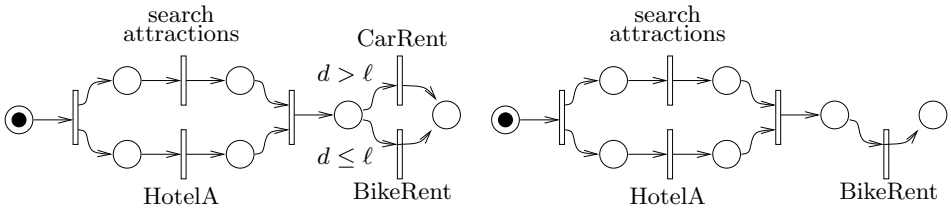

Fig. 2. The Travel Planner orchestration (left); a simplified version (right)

plans to visit along with the dates of his visit. The orchestration looks for a hotel in that city (service *HotelA*) for those dates and parallelly looks for sites of attractions (service *Search Attractions*) in the city. Once both these tasks are completed, it calculates the maximal distance 'd' between the hotel found and the attraction sites. If this distance is less than a certain threshold ℓ, a bike

rental service is called to get quotes for a rental bike. If distance d exceeds ℓ, then *Car Rent* is called to get quotes for a rental car instead. The orchestration to the right in Figure 2 is a simplified version of travel planner, in which it is assumed that all returns from HotelA are closer than ℓ to the attraction site.

This Travel Planner orchestration is monotonic: Increasing (or decreasing) the response time of any of its component services does result in a corresponding increase (or decrease) in the end to end latency. Monotonicity holds in this case because increasing (or decreasing) the response time of the services called first does not affect the value returned by these services.

The Travel Planner orchestration – A Modified Version The presence of timeouts and data dependant choices in orchestrations can however complicate things. Figure 3 (left) is a modified version of the Travel planner example where quotes for hotels are obtained from two services, *HotelA* and *HotelB*. Such an extension is quite natural in orchestrations, where a pool of services with similar functionality are queried with the same request. The orchestration selects the best response obtained from the pool, or combines their responses. In this modified Travel Planner example, of the two hotel offers received, the cheaper one is taken. Calls to the hotels are guarded by timers: if only one hotel has replied before a timeout, the response of the other is ignored. The rest of the example is unchanged.

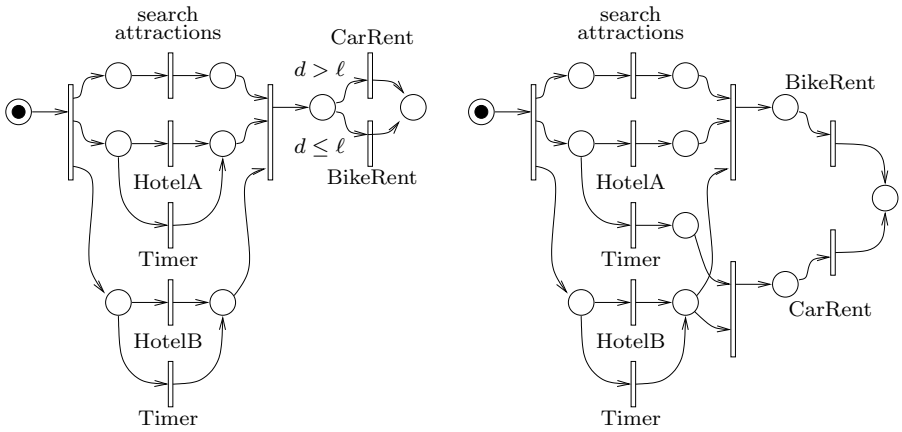

Fig. 3. The Modified Travel Planner orchestration. By convention, each *Timer* has priority over the *HotelX* service it is in conflict with. Left (a), right (b)

Now look at the following scenario: *HotelA* returns propositions that are usually cheaper than those of *HotelB* and so *HotelA*'s propositions are chosen. Let the distance d in this case be greater than ℓ and so service *Car Rent* is called. If the performance of *HotelA* now degrades such that it doesn't reply before a timeout, only *HotelB*'s response is taken. Say that the maximum distance d in

this case is less than ℓ and so service *Bike Rent* is called. Now if *Car Rent* takes a significantly greater time to respond compared to *Bike Rent*, it is possible that the overall latency is shorter in the second case. That is, a degradation in the performance of a service (*HotelA* here) leads to an improvement in the overall performance of the orchestration.

A solution to this is to make the choice in the Travel Planner orchestration dependent on the orchestration's client. For e.g, if we alter this orchestration such that the client specifies in the start of the orchestration whether he wants to rent a car or a bike, the choice is resolved by the client. The exact execution path of the orchestration is known at the start, on receiving the client's request. This execution path is a partial order, which is monotonic. We could then have input-dependent contracts, e.g., promising a certain response time for a given set of input parameters and promising another response behaviour for a different set of inputs.

The orchestration to the right in Figure 3 assumes that HotelA's propositions are all close to the attraction sites, whereas those of HotelB are all far from them. The net on the left can thus be simplified to the guard-free net of the right.

The examples in figure 3 are non-monotonic due to the presence of choice followed by paths with different performances. In the sequel, we formally characterize the classes of orchestrations that are monotonic, giving both necessary and sufficient conditions for it. The formal material for this is introduced next.

3 The Orchestration Model: OrchNets

In this section we present the high level Petri Nets model for orchestrations that we use for our studies, which we call *OrchNets*. OrchNets are a special form of *colored occurrence nets (CO-nets)*.

We have chosen this mathematical model for the following reasons. From the semantic studies performed for BPEL [9, 2] and Orc [6, 12], we know that we need to support in an elegant and succinct way the following features: concurrency, rich control patterns including preemption, representing data values, and for some cases even recursion. The first three requirements suggest using colored Petri nets. The last requirement suggests considering extensions of Petri nets with dynamicity. However, in our study we will not be interested in the specification of orchestrations, but rather in their executions. Occurrence nets are concurrent models of executions of Petri nets. As such, they encompass orchestrations involving recursion at no additional cost. The executions of Workflow Nets [13] are also CO-nets.

3.1 Background on Petri nets and Occurrence nets

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*.

The elements in $\mathcal{P} \cup \mathcal{T}$ are called the *nodes* of N and will be denoted by variables for *e.g*, x. For a node $x \in \mathcal{P} \cup \mathcal{T}$, 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. A *marking* of the net is a multiset M of places, *i.e* a map from $\mathcal{P}$ to $\mathbf{N}$. A transition t is *enabled* in marking M if $\forall p \in {}^\bullet t, M(p) > 0$. This enabled transition can *fire* resulting in a new marking $M - {}^\bullet t + t^\bullet$ denoted by $M[t\rangle M'$. A marking M is *reachable* if there exists a sequence of transitions $t_0, t_1 \dots t_n$ such that $M_0[t_0\rangle M_1[t_1\rangle \dots [t_n\rangle M$. A net is *safe* if for all reachable markings M, $M(\mathcal{P}) \subseteq \{0,1\}$.

For a net $N = (\mathcal{P}, \mathcal{T}, \mathcal{F}, M_0)$ the *causality relation* $<$ is the transitive closure of the flow relation $\mathcal{F}$. The reflexive closure of $<$ is denoted by $\leq$. 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* - written as $x \| y$ - if neither $(x \leq y)$ nor $(y \leq x)$ nor $(x \# y)$. A set of concurrent places $P \subseteq \mathcal{P}$ is called a *co-set*. A *cut* is a maximal (for set inclusion) co-set.

A *configuration* of N is a subnet κ of nodes of N such that:

1. κ is *causally closed*, *i.e*, if $x < x'$ and $x' \in \kappa$ then $x \in \kappa$
2. κ is *conflict-free*, *i.e*, for all nodes $x, x' \in \kappa, \neg(x \# x')$

For convenience, we will assume that the maximal nodes (w.r.t the $<$ relation) in a configuration are places.

A safe net $N = (\mathcal{P}, \mathcal{T}, \mathcal{F}, M_0)$ is called 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. $M_0 = \{p \in \mathcal{P} | {}^\bullet p = \emptyset\}$, *i.e* the initial marking is the set of minimal places with respect to $\leq$.

Occurrence nets are a good model for representing the possible executions of a concurrent system. *Unfoldings* of a safe Petri net, which collect all the possible executions of the net, are occurrence nets. Unfoldings are defined as follows. For N and N' two safe nets, a map $\varphi : \mathcal{P} \cup \mathcal{T} \mapsto \mathcal{P}' \cup \mathcal{T}'$ is called a *morphism* of N to N' if: 1/ $\varphi(\mathcal{P}) \subseteq \mathcal{P}'$ and $\varphi(\mathcal{T}) \subseteq \mathcal{T}'$, and 2/ for every $t \in \mathcal{T}$ and $t' = \varphi(t) \in \mathcal{T}'$, ${}^\bullet t \cup \{t\} \cup t^\bullet$ is in bijection with ${}^\bullet t' \cup \{t'\} \cup t'^\bullet$ through φ. A *branching process* of a safe net N is a pair (U, φ) where U is an occurrence net and $\varphi : U \mapsto N$ is a morphism such that 1/ φ establishes a bijection between M_0 and the minimal places of U, and 2/ ${}^\bullet t = {}^\bullet t'$ and $\varphi(t) = \varphi(t')$ together imply $t = t'$. Branching processes are partially ordered (up to isomorphism) by the prefix order and there exists a unique maximal branching process called the *unfolding* of N and denoted by U_N. The configurations of U_N capture the executions of N, seen as partial orders of events. For a configuration κ of an occurrence net N, the *future* of κ in N, denoted by N^κ is a sub-net of N with the nodes:

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

where $max(\kappa)$ is the set of maximal nodes of κ (which are all places by our restriction on configurations).

3.2 Orchestration model: OrchNets

We now present the orchestration model that we use for our studies, which we call *OrchNets*. OrchNets are occurrence nets in which

tokens are equipped with a special attribute, referred to as a *color,* and consisting of a pair (value, date). (1)

Figure 4 shows an OrchNet with its dates. Each place is labeled with a date

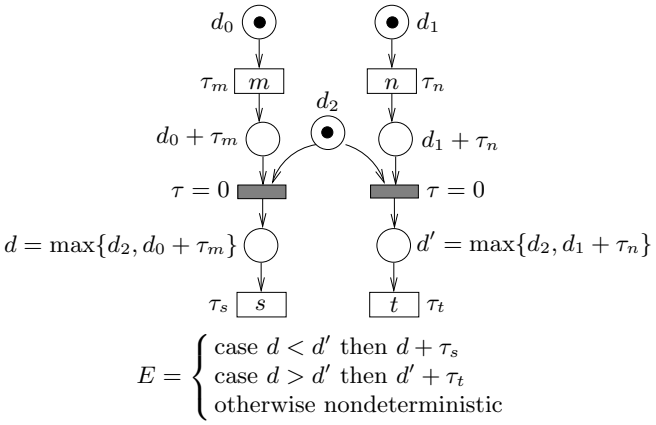

Fig. 4. An OrchNet showing the dates of its tokens. The delay of a transition is shown next to it.

which is the date of the token on reaching that place. Transitions are labeled with latencies. The tokens in the three minimal places are given *initial dates* (here, d_0, d_1, d_2). The four named transitions m, n, s and t are labeled with *latencies* τ_m, τ_n, τ_s and τ_t respectively, and the two shaded transitions have zero latency.

The presence of dates in tokens alters the firing semantics. A transition t is enabled at a date when all places in its preset have tokens. and if its guard evaluates to *true* (absence of a guard is interpreted as the guard *true*). Once enabled, transition t takes τ_t additional time to fire. For example, the shaded transition in the left has all its input tokens at $\max\{d_2, d_0+\tau_m\}$ and so it fires at $\max\{d_2, d_0+\tau_m\}+0$ since it has zero latency. If a transition fires at date d, then the tokens in its postset have the date d. This is shown in the figure, e.g., on the place following the left shaded transition, which has date $\max\{d_2, d_0 + \tau_m\}$.

When transitions are in conflict, (e.g., the two shaded transitions in Figure 4), the transition that actually occurs is governed by a *race policy* [4, 8]. If a set of enabled transitions are in conflict, the one with smallest date of occurrence will fire, preempting the other transitions in conflict with it. In Figure 4, the left or the right shaded transition will fire depending on whether $d < d'$ or $d > d'$ respectively, with a nondeterministic choice if $d = d'$. This results in selecting

the left most or right most continuation (firing s or t) accordingly. The resulting overall latency E of the orchestration is shown at the bottom of the figure.

In addition to dates, tokens in OrchNets can have data attributes, which we call *values*. We have not shown this in Figure 4, in order to keep it simple. Values of tokens in the preset of a transition t can be combined by a value function ϕ_t attached to t. The resulting value is taken by the token in the postset of t. At this point we are ready to provide the formal definition of *OrchNets*:

Definition 1 (OrchNet). *An* OrchNet *is a tuple* $\mathcal{N} = (N, \Phi, T, T_{\text{init}})$ *consisting of*

- *An occurrence net* N *with token attributes* $c = (value, date)$.
- *A family* $\Phi = (\phi_t)_{t \in \mathcal{T}}$ *of* value functions, *whose inputs are the values of the transition's input tokens.*
- *A family* $T = (\tau_t)_{t \in \mathcal{T}}$ *of* latency functions, *whose inputs are the values of the transition's input tokens.*
- *A family* $T_{\text{init}} = (\tau_p)_{p \in \min(\mathcal{P})}$ *of* initial date functions *for the minimal places of* N.

In general, value, latency, and initial date 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 a given value ω of this daemon, $\phi_t(\omega)$, $\tau_t(\omega)$, and $\tau_p(\omega)$ are all deterministic functions of their respective inputs.

3.3 The semantics of OrchNets

We now explain how the presence of dates attached to tokens affects the semantics of OrchNets by adopting the so-called *race policy*. We first describe how a transition t modifies the attributes of tokens. Let the preset of t have n places whose tokens have $(value, date)$ attributes $(v_1, d_1) \dots (v_n, d_n)$. Then all the tokens in the postset of t have the pair (v_t, d_t) of value and date, where:

$$\begin{aligned} v_t &= \phi_t(v_1 \dots v_n) \\ d_t &= \max\{d_1 \dots d_n\} + \tau_t(v_1 \dots v_n) \end{aligned} \tag{2}$$

The *race policy* for firing transitions is as follows. In any given marking M, let T be the set of transitions that are *possibly enabled*, i.e. $\forall t \in T$, ${}^\bullet t$ is marked in M and the guard of t (if any) is true. Then the transition t that is *actually enabled*, (which really fires) is given by:

$$t = \arg\min_{t \in T} d_t,$$
$$\text{where: } \arg\min_{x \in X} f(x) = x^* \in X \text{ s.t. } \forall x' \in X, f(x^*) \leq f(x').$$

If two possibly enabled transitions have the same d_t, then the choice of the transition that actually fires is non-deterministic. The race policy has the effect of filtering out configurations of OrchNets as explained now. Let $\mathcal{N} = (N, \Phi, T, T_{\text{init}})$

be a finite OrchNet. For a value $\omega \in \Omega$ for the daemon we can calculate the following *dates* for every transition t and place p of $\mathcal{N}$:

$$\begin{array}{l} d_p(\omega) = \tau_p(\omega) \textit{ if } p \textit{ is minimal, } d_s(\omega) \text{ where } s = {}^\bullet p \textit{ otherwise} \\ d_t(\omega) = \max\{d_x(\omega) \mid x \in {}^\bullet t\} + \tau_t(\omega)(v_1, \ldots v_n) \end{array} \tag{3}$$

where $v_1, \ldots v_n$ are the value components of the tokens in ${}^\bullet t$ as in equation (2). If κ is a configuration of N, the future $\mathcal{N}^\kappa$ is the OrchNet $(N^\kappa, \Phi_{N^\kappa}, T_{N^\kappa}, T'_{\text{init}})$ where Φ_{N^κ} and T_{N^κ} are the restrictions of Φ and T respectively, to the transitions of N^κ. T'_{init} is the family derived from $\mathcal{N}$ according to (3): for any minimal place p of N^κ, the initialisation function is given by $\tau'_p(\omega) = d_p(\omega)$. For a net N with the set of transitions $\mathcal{T}_N$, set $\mathcal{T}_{\min}(N) = \{t \in \mathcal{T}_N \mid {}^\bullet\!{}^\bullet t \cap \mathcal{T}_N = \emptyset\}$. Let $min(\mathcal{P}_N)$ denote the minimal places of N. Now define $\kappa_0(\omega) = min(\mathcal{P}_N)$ and inductively,

$$\begin{array}{c} \text{for } m > 0 : \kappa_m(\omega) = \kappa_{m-1}(\omega) \cup \{t_m\} \cup {}^\bullet t_m \cup t_m{}^\bullet \\ \text{where } t_m = \arg \min_{t \in \mathcal{T}_{\min}(N^{\kappa_{m-1}(\omega)})} d_t(\omega) \end{array} \tag{4}$$

Since net N is finite, the above inductive definition terminates in finitely many steps when $N^{\kappa_m(\omega)} = \emptyset$. Let $M(\omega)$ be this number of steps. We thus have

$$\emptyset = \kappa_0 \subset \kappa_1(\omega) \cdots \subset \kappa_{M(\omega)}(\omega)$$

$\kappa_{M(\omega)}(\omega)$ is a maximal configuration of $\mathcal{N}$ that can actually occur according to the race policy, for a given $\omega \in \Omega$; such actually occurring configurations are generically denoted by

$$\overline{\kappa}(\mathcal{N}, \omega)$$

For B, a prefix-closed subset of the nodes of N define

$$E_\omega(B, \mathcal{N}) = \max\{d_x(\omega) \mid x \in B\} \tag{5}$$

If B is a configuration, then $E_\omega(B, \mathcal{N})$ is the time taken for B to execute (latency of B). The latency of the OrchNet $\mathcal{N} = (N, \Phi, T, T_{\text{init}})$ for a given ω is

$$E_\omega(\mathcal{N}) = E_\omega(\overline{\kappa}(\mathcal{N}, \omega), \mathcal{N}) \tag{6}$$

Our design choices for the semantics of OrchNets were inspired by the application domain, i.e. compositions of web services. They reflect the following facts:

- Since we focus on latency, $\{\text{value}, \text{date}\}$ is the only color needed.
- Orchestrations rarely involve decisions on actions based on absolute dates. Timeouts are an exception, but these can be modelled explicitly, without using dates in guards of transitions. This justifies the fact that guards only have token values as inputs, and not their dates.
- The time needed to perform transitions does not depend on the tuple of dates $(d_1 \ldots d_n)$ when input tokens were created, but it can depend on the data $(v_1 \ldots v_n)$ and computation ϕ performed on these. This justifies our restriction for output arc expressions.

If it is still wished that control explicitly depends on dates, then dates must be measured and can then be stored as part of the value v.

4 Characterizing monotonicity

In this article, we are interested in the total time taken to execute a web-service orchestration. As a consequence, we will consider only orchestrations that terminate in a finite time, *i.e*, only a finite number of values can be returned.

4.1 Defining and characterizing monotonicity

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

Definition 2 (pre-OrchNet). *Call* pre-OrchNet *a tuple* $\mathbb{N} = (N, \Phi, \mathbb{T}, \mathbb{T}_{\text{init}})$*, where* N *and* Φ *are as before, and* $\mathbb{T}$ *and* $\mathbb{T}_{\text{init}}$ *are sets of families* T *of latency functions and of families* T_{init} *of initial date functions. Write* $\mathcal{N} \in \mathbb{N}$ *if* $\mathcal{N} = (N, \Phi, T, T_{\text{init}})$ *for some* $T \in \mathbb{T}$ *and* $T_{\text{init}} \in \mathbb{T}_{\text{init}}$*.*

For two families T and T' of latency functions, write

$$T \geq T'$$

to mean that $\forall\omega \in \Omega, \forall t \in \mathcal{T} \implies \tau_t(\omega) \geq \tau'_t(\omega)$, and similarly for $T_{\text{init}} \geq T'_{\text{init}}$. For $\mathcal{N}, \mathcal{N}' \in \mathbb{N}$, write

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

to mean that $T \geq T'$ and $T_{\text{init}} \geq T'_{\text{init}}$ both hold, and $E_\omega(\mathcal{N}) \geq E_\omega(\mathcal{N}')$ holds for every ω, respectively.

Definition 3 (monotonicity). *pre-OrchNet* $\mathbb{N} = (N, \Phi, \mathbb{T}, \mathbb{T}_{\text{init}})$ *is called* monotonic *if, for any two* $\mathcal{N}, \mathcal{N}' \in \mathbb{N}$*, such that* $\mathcal{N} \geq \mathcal{N}'$*, we have* $E(\mathcal{N}) \geq E(\mathcal{N}')$*.*

Theorem 1 (a global necessary and sufficient condition).

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

$$\forall\mathcal{N} \in \mathbb{N}, \forall\omega \in \Omega, \forall\overline{\kappa} \in \overline{\mathcal{V}}\,(N) \implies E_\omega(\overline{\kappa}, \mathcal{N}) \geq E_\omega(\overline{\kappa}(\mathcal{N}, \omega), \mathcal{N}) \quad (7)$$

   where $\overline{\mathcal{V}}\,(N)$ *denotes the set of all maximal configurations of net* N *and* $\overline{\kappa}(\mathcal{N}, \omega)$ *is the maximal configuration of* $\mathcal{N}$ *that actually occurs under the daemon value* ω*.*
2. *Conversely, assume that:*
   (a) Condition (7) 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 $\mathbb{N} = (N, \Phi, \mathbb{T}, \mathbb{T}_{\text{init}})$ *is not monotonic.*

Statement 2 expresses that Condition (7) is also necessary provided that it is legal to increase at will latencies or initial dates. Observe that violating Condition (7) does not by itself cause non-monotonicity; as a counterexample, consider a case where $\mathbb{T}$ is a singleton for which (7) is violated—it is nevertheless monotonic.

The orchestration in the left of Figure 2 satisfies Theorem 1 trivially, since for any given ω, there is only one possible maximal configuration. This is because the value of d is fixed for a ω and only one branch of the two rental services is enabled. The orchestration in the left of Figure 3 does not fulfill Theorem 1. Consider an ω for which the actually occurring configuration κ has both the responses of *HotelA* and *HotelB*. Say that $d > \ell$ for κ and *Car Rent* is called. Now consider another configuration κ' (under the same ω), got by replacing *HotelA* by *Timer*. In this case, the response of *Hotel B* is used to calculate d, which may be different from that in configuration κ. This d could be less than ℓ causing *Bike Rent* to be called. In this case, the latencies of *Car Rent* and *Bike Rent* can be set such that $E_\omega(\kappa, \mathcal{N}) > E_\omega(\kappa', \mathcal{N})$, violating Theorem 1.

4.2 A structural condition for the monotonicity of workflow nets

Workflow nets [13] 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* [13]. 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, this yields *pre-WFnets* $\mathbb{W} = (W, \Phi, \mathbb{T}, \mathbb{T}_{\text{init}})$. Referring to the end of Section 3.1, 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

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

by attaching to each transition t of N_W the value and latency 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

latency 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* [8] on nets. For a net N, a *cluster* is a (non-empty) 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 2 (Sufficient Condition). *Let W be a WFnet and N_W be its unfolding. A sufficient condition for the pre-OrchNet $\mathbb{N}_W = (N_W, \Phi_W, \mathbb{T}_W, \mathbb{T}_{\text{init}})$ to be monotonic is that every cluster $\mathbf{c}$ satisfies the following condition:*

$$\forall t_1, t_2 \in \mathbf{c},\ t_1 \neq t_2 \implies {t_1}^\bullet = {t_2}^\bullet \tag{8}$$

Recall that the sufficient condition for monotonicity stated in Theorem 1 is "almost necessary" in that, if enough flexibility exist in setting latencies and initial dates, then it is actually necessary. The same holds for the sufficient condition stated in Theorem 2 if the workflow net is assumed to be live.

Theorem 3 (Necessary Condition). *Suppose 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 latencies and initial dates. In addition, assume that there is at least one $\mathcal{W}^* \in \mathbb{W}$ such that there is an daemon value ω^* for which the latencies of all the transitions are finite. Then the sufficient condition of Theorem 2 is also necessary for monotonicity.*

Observe that the orchestration in the right of figure 2 satisfies Theorem 2, whereas the orchestration in the right of figure 3 does not.

5 Probabilistic monotonicity

So far we have considered the case where latencies of transitions are nondeterministic. In a previous work [10, 11], on the basis of experiments performed on real Web services, we have advocated the use of probability distributions when modeling the response time of a Web service. Can we adapt our theory to encompass probabilistic latencies?

5.1 Probabilistic setting, first attempt

In Definitions 1 and 2, latency and initial date functions were considered nondeterministic. The first idea is to let them become random instead. This leads to the following straightforward modification of definitions 1 and 2:

Definition 4 (probabilistic OrchNet and pre-OrchNet, 1). *Call* probabilistic OrchNet *a tuple $\mathcal{N} = (N, \Phi, T, T_{\text{init}})$ where $\Phi = (\phi_t)_{t \in \mathcal{T}}$, $T = (\tau_t)_{t \in \mathcal{T}}$, and $T_{\text{init}} = (\tau_p)_{p \in \min(\mathcal{P})}$, are independent families of* random *value functions, latency functions, and initial date functions, respectively.*

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

We now equip random latencies and initial dates with a probabilistic ordering. If τ is a random latency function, its *distribution function* is defined by

$$F(x) = \mathbf{P}(\tau \leq x)$$

where $x \in \mathbb{R}_+$. Consider the following ordering: random latencies τ and τ' satisfy

$$\tau \geq_s \tau' \text{ if } F(x) \leq F'(x) \text{ holds } \forall x \in \mathbb{R}_+, \tag{9}$$

where F and F' are the distribution functions of τ and τ', respectively—with corresponding definition for the probabilistic ordering on initial date functions. Order (9) is classical in probability theory, where it is referred to as *stochastic dominance* or *stochastic ordering* among random variables [1].

Using order (9), for two families T and T' of random latency functions, write

$$T \geq_s T'$$

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

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

if $T \geq_s T'$ and $T_{\text{init}} \geq_s T'_{\text{init}}$ 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 5 (probabilistic monotonicity, 1). *Probabilistic pre-OrchNet* $\mathbb{N}$ *is called* probabilistically monotonic *if, for any 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 3 implies probabilistic monotonicity in the sense of to definition 5. 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.

5.2 Probabilistic setting: second attempt

Let us restart from the nondeterministic setting of sections 3 and 4. Focus on definition 1 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 latencies and initial dates random. This suggests the following alternative setting for probabilistic monotonicity.

Definition 6 (probabilistic OrchNet and pre-OrchNet, 2). *Call* probabilistic OrchNet *a pair* $(\mathcal{N}, \mathbf{P})$*, where* $\mathcal{N}$ *is an OrchNet according to definition 1 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 2 and* $\mathbf{P}$ *is a probability over the domain* Ω *of all values for the daemon.*

How can we relate the two definitions 4 and 6? Consider the following assumption, which will be in force in the sequel:

Assumption 1 *For any* $\mathcal{N} \in \mathbb{N}$*,* τ_t *and* τ_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 4. For t a generic transition, let $(\Omega_t, \mathbf{P}_t)$ be the set of possible experiments together with associated probability, for random latency τ_t; and similarly for $(\Omega_p, \mathbf{P}_p)$ and τ_p. Thanks to assumption 1, setting

$$\Omega = (\prod_t \Omega_t) \times (\prod_p \Omega_p) \quad \text{and} \quad \mathbf{P} = (\prod_t \mathbf{P}_t) \times (\prod_p \mathbf{P}_p), \tag{10}$$

yields the entities of definition 6. 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 2, we said that

$$\tau \geq \tau' \text{ if } \tau(\omega) \geq \tau'(\omega) \text{ holds } \forall \omega \in \Omega, \tag{11}$$

Clearly, if two random latencies τ and τ' satisfy condition (11), then they also satisfy condition (9). That is, *ordering* (11) *is stronger than stochastic ordering* (9). Unfortunately, the converse is *not* true in general. For example, condition (9) may hold while τ and τ' are two independent random variables, which prevents (11) from being satisfied. Nevertheless, the following routine result holds:

Theorem 4. *If condition* (9) *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* $\hat{\tau}$ *and* $\hat{\tau}'$ *over* Ω*, such that:*

1. $\hat{\tau}$ *and* $\hat{\tau}'$ *possess* F *and* F' *as respective distribution functions, and*
2. *condition* (11) *is satisfied by the pair* $(\hat{\tau}, \hat{\tau}')$ *with probability* 1*.*

Proof. Take $\Omega = [0, 1]$ and $\mathbf{P}$ the Lebesgue measure. Then, taking, $\hat{\tau}(\omega) = \inf\{x \in \mathbb{R}_+ | F(x) \geq \omega\}$ and $\hat{\tau}'(\omega) = \inf\{x \in \mathbb{R}_+ | F'(x) \geq \omega\}$ yields the claim.

Theorem 4 allows reducing the stochastic comparison of real valued 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 latency function and each random initial date function, when considered in isolation. Thus, when performing construction (10) for two OrchNets $\mathcal{N}$ and

$\mathcal{N}'$, we can take the *same* pair $(\Omega_t, \mathbf{P}_t)$ to represent both τ_t and τ'_t, and similarly for τ_p and τ'_p. Applying (10) implies that both $\mathcal{N}$ and $\mathcal{N}'$ are represented using the *same* pair $(\Omega, \mathbf{P})$. This leads naturally to definition 6.

In addition, applying theorem 4 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 4 requires. Thus we can reformulate probabilistic monotonicity as follows—compare with definition 5:

Definition 7 (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 5. *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 $\tau \geq \tau'$ implies $\tau \geq_s \tau'$ if τ and τ' 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 1:

Theorem 6. *Assume condition 2b of theorem 1 is satisfied. Then, if probabilistic pre-OrchNet* $(\mathbb{N}, \mathbf{P})$ *is probabilistically monotonic, then it is also monotonic with* $\mathbf{P}$*-probability* 1*.*

6 Getting Rid of Non-Monotonicity

Avoiding Non-Monotonicity. We suggest a few ways in which non-monotonic orchestrations can be made monotonic. These might serve as guidelines to the designer of an orchestration, to avoid building non-monotonic orchestrations.

1. *Eliminate Choices.* We saw that choices in the execution flow can create non-monotonicity. So if possible, choices in the execution flow should be avoided while designing orchestrations. This seems very restrictive but is not totally unrealistic. For example, in the Travel Planner orchestration of figure 3, if the designer can find a rental service for both, cars and bikes, then the two mutually exclusive rental calls can be replaced by a call to that single rental service. This makes the execution flow an event graph and the Travel Planner orchestration monotonic.
2. *Balancing out performance of mutually exclusive branches.* One way to make an orchestration "more monotonic" is to ensure that all its mutually exclusive branches have similar response times. For e.g., in the Travel Planner example

of figure 3, if the two exclusive services *Bike Rent* and *Car Rent* have similar response times, the orchestration is nearly monotonic.

3. *Externalising Choices.* Choices are of course integral to many execution flows and sometimes simply cannot be removed. A possible way out in this case is to *externalise* the choice and make them client dependent. This solution has already been discussed in the modified Travel Planner example of Section 2.
4. *If none of the above works,* then a brute force alternative consists in performing the following. Replace the orchestration latency $E_\omega(\mathcal{N})$ defined in (6) by the following pessimistic bound for it (see Theorem 1 for the notations):

$$F_\omega(\mathcal{N}) = \max\left\{E_\omega(\kappa, \mathcal{N}) \mid \overline{\kappa} \in \overline{\mathcal{V}}\,(N)\right\} \tag{12}$$

Then for any net N, and any two OrchNets $\mathcal{N}$ and $\mathcal{N}'$ over N, $\forall\omega$

$$F_\omega(\mathcal{N}) \geq E_\omega(\mathcal{N}) \tag{13}$$

$$\mathcal{N} \geq \mathcal{N}' \Rightarrow F_\omega(\mathcal{N}) \geq F_\omega(\mathcal{N}') \tag{14}$$

holds. Therefore, using the pessimistic bound $F_\omega(\mathcal{N})$ instead of tight estimate $E_\omega(\mathcal{N})$ when building the orchestration's contract with its customer, is safe in that: 1) by (14), monotonicity of $F_\omega(\mathcal{N})$ with respect to the underlying OrchNet is guaranteed, and 2) by (13), the orchestration will meet its contract if its sub-contractors do so. In turn, this way of composing contracts is pessimistic and should therefore be avoided whenever possible.

Where does monotonicity play a role in the orchestration's life cycle? We use contracts to abstract the behaviour of the services involved in an orchestration. The orchestration, trusting these contracts, composes them to derive an estimate of its own performance, from which a contract between the orchestration and its customers can be established. Since this relies on trust between the orchestration and its sub-contractors, these contracts will have to be monitored at run-time to make sure that the sub-contractors deliver the promised performance. In case of violation, counter-measures like reconfiguring the orchestration might be taken. The orchestration's life cycle thus consists of the following phases [11]:

1. At *design time*, establish QoS contracts with the customer by composing QoS contracts from the called services; tune the monitoring algorithms accordingly; design reconfiguration strategy.
2. At *run time*, run the orchestration; in parallel, monitor the called services for possible QoS contract violation; whenever needed, perform reconfiguration.

Monotonicity plays a critical role at design time. The above pessimistic approach can be used as a backup solution if monotonicity is not satisfied. Monotonicity is however, not an issue at run time and the orchestration can be taken as such, with no modification. Monitoring of the called services remains unchanged too.

7 Conclusion

This paper is a contribution to the fundamentals of contract based QoS management of Web services orchestrations. QoS contracts implicitly assume monotonicity w.r.t. QoS parameters. We focus on one representative QoS parameter,

namely response time. We have shown that monotonicity is easily violated in realistic cases. We have formalized monotonicity and have provided necessary and sufficient conditions for it. As we have seen, QoS can be very often traded for Quality of Data: poor quality responses to queries (including exceptions or invalid responses) can often be got much faster. This reveals that QoS parameters should not be considered separately, in isolation. We have provided guidelines for getting rid of non-monotonicity.

We see one relevant extension of this work: Advanced orchestration languages like Orc [7] offer a sophisticated form of preemption that are modelled by contextual nets (with read arcs). Our mathematical results do not consider nets with read arcs. Extending our results to this case would be interesting and useful.

References

1. Gordon Anderson. Nonparametric tests of stochastic dominance in income distributions. *Econometrica*, 64(5):1183–93, September 1996.
2. Jesús Arias-Fisteus, Luis Sánchez Fernández, and Carlos D. Kloos. Applying model checking to BPEL4WS business collaborations. In *SAC*, pages 826–830, 2005.
3. OASIS WSBPEL Technical Committee. Web Services Business Process Execution Language Version 2.0. OASIS Standard, April 2007.
4. D. Kartson, G. Balbo, S. Donatelli, G. Franceschinis, and G. Conte. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, Inc., NY, USA, 1994.
5. A. Keller and H. Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *J. Network Syst. Manage.*, 11(1), 2003.
6. David Kitchin, William R. Cook, and Jayadev Misra. A language for task orchestration and its semantic properties. In *CONCUR*, pages 477–491, 2006.
7. 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`.
8. Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.
9. Chun Ouyang, Eric Verbeek, Wil M. P. van der Aalst, Stephan Breutel, Marlon Dumas, and Arthur H. M. ter Hofstede. Formal semantics and analysis of control flow in ws-bpel. *Sci. Comput. Program.*, 67(2-3):162–198, 2007.
10. 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.
11. Sidney Rosario, Albert Benveniste, Stefan Haar, and Claude Jard. Probabilistic QoS and Soft Contracts for Transaction based Web Services. *IEEE Transactions on Services Computing*, 1(4):187–200, 2008.
12. Sidney Rosario, David Kitchin, Albert Benveniste, William R. Cook, Stefan Haar, and Claude Jard. Event structure semantics of orc. In *WS-FM*, pages 154–168, 2007.
13. Wil M. P. van der Aalst. Verification of workflow nets. In *ICATPN*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer, 1997.
14. 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 Proof of Theorem 1

Proof. We first prove Statement 1. Let $\mathcal{N}' \in \mathbb{N}$ be such that $\mathcal{N}' \geq \mathcal{N}$. We have:

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

where the first inequality follows from the fact that $\overline{\kappa}(\mathcal{N}', \omega)$ is a conflict free partial order and $\mathcal{N}' \geq \mathcal{N}$, and the second inequality follows from (7) applied with $\overline{\kappa} = \overline{\kappa}(\mathcal{N}', \omega)$. This proves Statement 1.

We prove statement 2 by contradiction. Let $(\mathcal{N}, \omega, \overline{\kappa}^\dagger)$ be a triple violating Condition (7), in that

$$\overline{\kappa}^\dagger \text{ cannot occur, but } E_\omega(\overline{\kappa}^\dagger, \mathcal{N}) < E_\omega(\overline{\kappa}(\mathcal{N}, \omega), \mathcal{N}) \text{ nevertheless holds.}$$

Now consider the OrchNet net $\mathcal{N}' = (N, \Phi, T', T_{\text{init}})$ where the family T' is the same as T except that in ω, $\forall t \notin \overline{\kappa}^\dagger$, $\tau'_t(\omega) > E_\omega(\overline{\kappa}^\dagger, \mathcal{N})$. Clearly $\mathcal{N}' \geq \mathcal{N}$. But using construction (4), it is easy to verify that $\overline{\kappa}(\mathcal{N}', \omega) = \overline{\kappa}^\dagger$ and thus

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

which violates monotonicity.

A.2 Proof of Theorem 2

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 1 holds for $\mathcal{N}$ by induction on the number of transitions in the maximal configuration $\overline{\kappa}(\mathcal{N}, \omega)$ that actually occurs. The base case is when it has only one transition. Clearly this transition has the least latency and any other maximal configuration has a greater execution time.

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

$$E_\omega(\overline{\kappa}, \mathcal{N}) \geq E_\omega(\overline{\kappa}(\mathcal{N}, \omega), \mathcal{N}) \tag{15}$$

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

Induction Argument. Consider the OrchNet $\mathcal{N}$, where the actually occurring configuration $\overline{\kappa}(\mathcal{N}, \omega)$ has m transitions. κ' is any other maximal configuration of $\mathcal{N}$. If the transition t in $\overline{\kappa}(\mathcal{N}, \omega)$ with minimal date d_t also occurs in κ' then comparing execution times of $\overline{\kappa}(\mathcal{N}, \omega)$ and κ' reduces to comparing $E_\omega(\overline{\kappa}(\mathcal{N}, \omega) \backslash \{t\}, \mathcal{N}^t)$ and $E_\omega(\kappa' \setminus \{t\}, \mathcal{N}^t)$. Since $\overline{\kappa}(\mathcal{N}, \omega) \setminus \{t\}$ is the actually occurring configuration in the future $\mathcal{N}^t$ of transition t, using our induction hypothesis, we have

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

and so

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

If $t \notin \kappa'$ for some κ', 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 $\mathbf{c}$. Hence, $t^\bullet = t'^\bullet$ follows from condition 8 of Theorem 2. 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 T_{init} and T'_{init} are the initial marking of these places, $T_{init} \leq T'_{init}$ (since $d_t \leq d_{t'}$, $t^\bullet$ has dates lesser than $t'^\bullet$). Hence

$$E_\omega(\overline{\kappa}(\mathcal{N},\omega),\mathcal{N}) \ = E_\omega(\overline{\kappa}(\mathcal{N},\omega) \setminus \{t\},\mathcal{N}^t) \tag{16}$$

and

$$E_\omega(\kappa',\mathcal{N}) \ = E_\omega(\kappa' \setminus \{t'\},\mathcal{N}^{t'}) \geq E_\omega(\kappa' \setminus \{t'\},\mathcal{N}^t) \tag{17}$$

The inequality holds since $\mathcal{N}^{t'} \geq \mathcal{N}^t$. The induction hypothesis on (16) and (17) gives $E_\omega(\overline{\kappa}(\mathcal{N},\omega),\mathcal{N}) \ \leq E_\omega(\kappa',\mathcal{N})$. This proves the theorem.

A.3 Proof of Theorem 3

Proof. We will show that when condition (8) of Theorem 2 is not satisfied by W, the Orchnets in its induced preOrchNet $\mathbb{N}_W$ can violate condition (7) of Theorem 1, the necessary condition for monotonicity.

Let c_W be any cluster in W that violates the condition 8 of Theorem 2. 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 $K = [t_1] \cup [t_2]$. We consider the following two cases:

K is a configuration. If so, consider the OrchNet $\mathcal{N}^* \in \mathbb{N}_W$ obtained when transitions of N_W (and so W) have latencies as that in $\mathcal{W}^*$. So for the daemon value ω^*, the quantity $E_{\omega^*}(K,\mathcal{N}^*)$ is some finite value $\mathbf{n}^*$. Now, configuration K can actually occur in a OrchNet $\mathcal{N}$, such that $\mathcal{N} > \mathcal{N}^*$, where $\mathcal{N}$ is obtained as follows (τ and τ^* denote the latencies of transitions in $\mathcal{N}$ and $\mathcal{N}^*$ respectively): $\forall t \in K, t' \in N_W$ s.t. ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$, set $\tau_{t'}(\omega^*) = \mathbf{n}^* + 1$ and keep the other latencies unchanged. In this case, for the daemon value ω^*, the latencies of all transitions of $\mathcal{N}$ (and so its overall execution time) is finite. Denote by $\mathcal{N}^K$ the future of $\mathcal{N}$ once configuration K has actually occurred. Both t_1 and t_2 are minimal and enabled in $\mathcal{N}^K$.

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 $\mathcal{N}^K$ 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: $\tau'_{t_1}(\omega^*) =$

$\tau_{t_1}(\omega^*), \tau'_{t_2}(\omega^*) = \tau_{t_1}(\omega^*) + 1$ and for all other $t \in c, \tau'_t(\omega^*) = \tau'_{t_2}(\omega^*) + 1$. Set $\tau'_{t^*}(\omega^*) = \infty$ and for all other transitions of $\mathcal{N}'$, the delays are the same as that in $\mathcal{N}$ and thus are finite for ω^*.

t_1 has the minimal delay among all transitions in c, and t^* is in the future of t_1. So the actually occurring configuration $E_{\omega^*}(\overline{\kappa}(\mathcal{N}', \omega^*), \mathcal{N}')$ has an infinite delay. However any maximal configuration $\overline{\kappa}$ which does not include t_1 (for eg, when t_2 fires instead of t_1) will have a finite delay. For such $\overline{\kappa}$ we thus have $E_{\omega^*}(\overline{\kappa}(\mathcal{N}', \omega^*), \mathcal{N}') > E_{\omega^*}(\overline{\kappa}, \mathcal{N}')$ and so $\mathcal{N}'$ violates the condition (7) of Theorem 1.

K 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 8 of Theorem 2 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 K being a configuration.

A.4 Proof of theorem 6

Proof. The proof is by contradiction. Assume that $\mathbb{N}$ is *not* monotonic with positive $\mathbf{P}$-probability, i.e., :

$$\begin{array}{l} \text{there exists a pair } (\mathcal{N}, \mathcal{N}') \text{ of OrchNets such that} \\ \mathcal{N} \geq \mathcal{N}' \text{ and } \mathbf{P}\{\, \omega \in \Omega \mid E_\omega(\mathcal{N}) < E_\omega(\mathcal{N}')\} > 0. \end{array} \tag{18}$$

To prove the theorem it is enough to prove that (18) implies:

$$\begin{array}{l} \text{there exists } \mathcal{N}_o, \mathcal{N}'_o \in \mathbb{N} \text{ such that } \mathcal{N}_o \geq \mathcal{N}'_o, \\ \text{but } E(\mathcal{N}_o) \geq_s E(\mathcal{N}'_o) \text{ does not hold} \end{array} \tag{19}$$

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 1 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 (18), 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.