Application of branching cells to QoS aware service orchestrations
Albert Benveniste, Claude Jard, Samy Abbes

To cite this version:

HAL Id: hal-01158211
https://hal.archives-ouvertes.fr/hal-01158211v2
Submitted on 13 Jul 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Branching Cells for Asymmetric Event Structures

Albert Benveniste\textsuperscript{a}, Claude Jard\textsuperscript{b}, Samy Abbes\textsuperscript{c}

\textsuperscript{a}\textit{Inria Rennes}
Campus de Beaulieu
35 000 Rennes, France

\textsuperscript{b}\textit{LINA/Université de Nantes}
2, rue de la Houssinière
44 322 Nantes Cedex 3, France

\textsuperscript{c}\textit{PPS/Université Paris Diderot}
Bâtiment Sophie Germain
Avenue de France
75013 Paris, France

Abstract

By allowing service calls to be guarded by contexts, Asymmetric Event Structures (AES for short) and contextual nets are a convenient framework to model composite Web services or service orchestrations. We equip AES with QoS domains as a framework to capture a number of QoS metrics and their combination. We use the resulting model to formalize QoS-based late service binding in composite services. When subject to QoS-based late service binding, composite services may be non-monotonic with respect to QoS, meaning that strictly improving the QoS of a service may strictly decrease the end-to-end QoS of the composite service, an embarrassing feature for QoS-aware management; we study this issue. Branching cells of AES play a central role in this study.

Keywords:

1. Introduction

Workflow nets \cite{20, 21, 22} have been advocated as a concurrent model for workflows. Queries submitted to the workflow are captured by tokens circulating in the workflow net. These tokens are enriched with the data returned by the called services and, when exiting the workflow net, tokens carry the result of the workflow in response to the submitted query. Formally speaking, executions of workflow nets are conveniently captured by Prime Event Structures (PES for short) with causality and symmetric binary conflict \cite{14}, capturing the successive choices made between alternative continuations of the workflow execution.

It is, however, very natural in service orchestrations, to guard service calls by contexts, i.e., conditions on variables, states, or attributes, that are only checked but not modified or consumed. Indeed, rich workflow, orchestration, or choreography languages (such as BPEL \cite{19} or Orc \cite{12, 13}), use contexts in

\textit{Email addresses:} albert.benveniste@inria.fr (Albert Benveniste), claude.jard@univ-nantes.fr (Claude Jard), samy.abbes@univ-paris-diderot.fr (Samy Abbes)

\textit{Preprint submitted to Elsevier} 2013
addition to resources to guard service calls. The basic notions of causality and conflict offered by safe Petri nets and associated PES, do not encompass the notion of context. The simplest extension of Petri nets to handle contexts is by adding read arcs, giving raise to contextual nets and their associated framework of Asymmetric Event Structures (AES) [4]. Indeed, Orc concurrent semantics was defined in [18] using AES.

Since the world of services is open, several services often compete at offering equivalent functions, albeit under different implementations or with different Quality of Service (QoS). QoS metrics typically include timing performance, availability, security, reliability, vulnerability, cost, and variations thereof. Optimizing orchestrations for QoS has been the subject of a large literature, see for example [9, 3, 23, 25]. Part of this literature is devoted to off-line and static optimal selection of services among competing alternatives. While this is a relevant approach for a close world of predefined services, it is not appropriate for an open world where available services may not be known in advance when the composite service is defined. QoS based late service binding [10, 8] has proposed instead, which consists, for an orchestration, in dynamically selecting at run time, based on QoS, which service will actually be called to provide the due function at a given stage of the orchestration. This naturally gives raise to complex competitions between multiple threads of services that could share some services for calling, a typical source of confusion in nets, i.e., situations in which conflicts are not local to a single place, thus reflecting that choice is among alternatives involving competing groups of called services.

To address QoS based management of composite services, we developed in previous work a framework of soft probabilistic QoS contracts for service orchestrations [15, 16, 17]. In this framework, the QoS of each called service with its different dimensions, is considered probabilistic. A QoS contract thus consists of a probability distribution that each service exposes (as part of its Service Level Agreement, SLA). The service meets its obligations as long as it behaves at least as good as stated in its contract—here “at least as good as” refers to stochastic ordering of random variables, as widely used in econometrics. Statistical tests were proposed in [16] to monitor the possible violation of probabilistic QoS contracts and a probabilistic contract composition method was proposed in [15, 16] for the particular case of latency, and in [17] for the general QoS case. The issue of monotonicity was first pointed out in [16]. It may be, for an orchestration, that a service improving its QoS still results in a degradation for the end-to-end QoS of the orchestration, an embarrassing feature when using contracts or performing late service binding. Such orchestrations are called non-monotonic. The first study of monotonicity was reported in [6], for the particular case of latency and for orchestrations involving no confusion. Besides [6], the above references targeted Web services audience and, thus, the formal side of this work was not detailed.

In this paper we formally study QoS-based late service binding in service orchestrations, for orchestrations modeled as AES with arbitrary confusion and for general, possibly multi-dimensional, QoS.

In companion paper [1], choices in AES with confusion are studied. This development serves us in this paper as a basis for studying QoS aware management of service orchestrations with emphasis on late service binding. To discuss an example, however, we feel it more comfortable to use safe contextual nets.
Companion paper [1] recalls in its appendix how to systematically construct, for any safe contextual net, the AES of all its executions.

Running Example 1. Figure 1 shows a toy example that will serve us as a support throughout this paper. For this discussion we use the example in its net form (top of the figure). For the moment, also, we ignore the blue labels. Each query submitted to the orchestration is represented by a fresh token entering the input place $P_0$. Responses to this query are collected at the return store of the orchestration, consisting of the two output places $P_{1\infty}$ and $P_{2\infty}$. The reception of a query triggers the launching of two parallel threads, beginning at the places $P_1$ and $P_2$.

Thread starting at $P_2$ calls service $S_2$, shown by event $a_2$; the execution of service $S_2$ is not atomic; $S_2$ silently operates while the token sits in place $P_{21}$; when terminating, shown by event $b_2$, service $S_2$ outputs a token in place $P_{2\infty}$. The reception of a query triggers the launching of two parallel threads, beginning at the places $P_1$ and $P_2$.

Thread starting at $P_1$ can nondeterministically call service $S_{11}$; the associated service call is captured by event $a_{11}$; service $S_{11}$ outputs a token in place $P_{1\infty}$ when terminated. As an alternative, thread starting at $P_1$ can call service $S_{12}$; the associated service call is captured by event $a_{12}$; the execution of service $S_{12}$ is not atomic; $S_{12}$ silently operates while the token sits in place $P_{12}$ and
outputs a token in place $P_\infty$ when terminated; $S_{12}$, however, requires for its completion to know a context consisting of an attribute (say, an Id) held by the second thread $P_2$. This attribute may be consumed when terminating service $S_2$, shown by event $b_2$, thus preventing $S_{12}$ from terminating in this case (event $b_{12}$ cannot be produced). Note that the failure of sites to respond is not atypical in the world of services.

This orchestration involves two choices following places $P_1$ and $P_2$ and one synchronization at event $b_{12}$. The choice occurring in $P_1$ captures late service binding at run time and we wish to perform it by selecting the alternative offering best QoS.

□

To this end, we develop an algebraic formalization of QoS, modeling how QoS quanta associated with service calls contribute to the end-to-end QoS of the orchestration. Quite often, several metrics are jointly considered, making QoS multi-dimensional and, thus, possibly only partially ordered.

Running Example 2. Focus again on Figure 1 and consider now its blue labels. Each label is a pair [latency, security level], thus capturing a two-dimensional QoS. Latencies are ordered as usual, where smaller means better. Security levels are ordered as high<low, where, again, smaller means better. The two-dimensional QoS is ordered by the product order. The execution of the workflow of Figure 1 is controlled by this two-dimensional QoS.

□

Having done this, we can now explain how QoS-based late service binding is performed for the considered workflow.

Running Example 3. A query enters the orchestration with zero latency and high security level.

- The first transition launches the two threads and we assume it takes no time and has no impact on security level. Therefore, when entering places $P_1$ and $P_2$, the token keeps the same QoS attributes.

- Once the second thread is launched, $a_2$ fires, which causes a latency $d_{21}$ while keeping security level high. Then, $b_2$ fires and outputs at place $P_2\infty$ a response for the orchestration with QoS $q_{\infty}^2 = [d_{21} + d_{22}, high]$.

- Focus next on the first thread and the choice occurring at place $P_1$. Event $a_{11}$ causes an increment in QoS as indicated on the figure, and similarly for $a_{12}$.

  - Suppose that $d_{12}$≤$d_{11}$. Since high is preferred to low regarding security level, QoS-based late service binding favors $S_{12}$ over $S_{11}$, so the latter is not called. Hence, $a_{12}$ get fired and the token reaches place $P_{12}$ with latency $0 + d_{12} = d_{12}$ and security level max(high, high) = high. If its context is available (meaning that a token still sits at $P_2$), service $S_{12}$ terminates in zero time with no degradation of the security level, which results in a response of the orchestration with QoS $q_{\infty}^1 = [d_{12} + d_{12}', high]$. This occurs if $d_{12}$≤$d_{21}$. The response of the orchestration in this case consists of the two concurrent responses from $S_{12}$ and $S_{2}$ and the end-to-end QoS for the orchestration is

$$\bigvee\{q_{\infty}^1, q_{\infty}^2\} = [\max(d_{12} + d_{12}', d_{21} + d_{22}), \max(high, high)] , \quad (1)$$

4
i.e., the “worst” QoS of the two responses. If, however, the context of
\( b_{12} \) is not available (\( d_{12} > d_{21} \)), then \( S_{12} \) never terminates (the token is
blocked in \( P_{12} \)) and then only \( S_2 \) responds with a resulting end-to-end
QoS equal to \( q_2^2 \).

Suppose instead that \( d_{12} > d_{11} \) and security level of \( a_{12} \) is also low.
Then, QoS-based late service binds favors \( S_{11} \) over \( S_{12} \). \( S_{11} \) termin-
ates and outputs a response with QoS \([d_{11}, low]\). On the other hand,
\( S_2 \) performs as for the other case, which results in an end-to-end QoS
for the orchestration equal to

\[
\sqrt{\{q_1^1, q_2^2\}} = [\max(d_{12}, d_{21} + d_{11}), \max(high, low)] \tag{2}
\]

with \( \max(high, low) = low \).

- The situation is less straightforward if \( a_{12} \) has security level high but
\( d_{12} > d_{11} \), making the two QoS incomparable (with respect to the product
order). Two strategies are then possible. One possibility is to select
nondeterministically one of the two alternatives (since no one outperforms
the other). The other possibility is to change the order used in comparing
QoS: one could weight and then add the two metrics; or one could give
priority, say, to security over latency. For both cases we get a total order
for the two-dimensional QoS.

This gives a picture of how QoS controls executions of this workflow. □

Running Example 4. Let us come back to the original product order with
QoS situation \([d_{11}, low]\) for \( a_{11} \) and \([d_{12}, low]\) for \( a_{12} \), with \( d_{11} < d_{12} \), so that
\( S_{11} \) is preferred over \( S_{12} \). Suppose service \( S_{12} \) improves its latency by reducing
\( d_{12} \) below \( d_{11} \) while, still, leaving \( d_{11} < d_{12} + d_{11}^1 \). Then, due to QoS-based
late service binding, \( S_{12} \) gets selected and the end-to-end QoS is worsened, from
\([\max(d_{11}, d_{21} + d_{22}), low]\) to \([\max(d_{12} + d_{11}^1, d_{21} + d_{22}), low]\). Thus, an improve-
ment in the QoS of some service has resulted in a degradation in the end-to-end
QoS of the orchestration. □

Running example 4 illustrates the lack of monotonicity with respect to QoS,
in that a site improving its own QoS may result in a degradation of the end-
to-end QoS of the orchestration. Which counter-measure could be considered
to avoid this? Of course, making choices on the basis of end-to-end global
optimization of the QoS is monotonic by definition. It does not fit the area of
composite services, however, where on-line dynamic service binding is preferred.

Running Example 5. For our example, one may argue that deciding between
\( a_{11} \) and \( a_{12} \) is questionable; it would rather make sense to compare \( S_{11} \) with \( S_{12} \)
as a whole, whose QoS equals \([d_{12} + d_{11}^1, low]\). Mononicity would be recovered
after this aggregation, but at the price of postponing the decision. □

To allow for an understandable and agreed management of service orches-
trations, monotonicity is required when performing QoS based service binding,
negotiation and reconfiguration. The failure to be monotonic is not a pathologi-
cal situation for an orchestration, however. It is, indeed, often encountered. The
more so when orchestrations are specified using rich languages such as BPEL or
Orc. It is not uncommon that a failure to be monotonic is not easy identifying.
Despite the prevalence of on-line decisions and reconfigurations in the area of composite services, the issue of monotonicity has not been properly acknowledged in the literature. The only exceptions we are aware of are [2, 3, 24], where global versus local QoS optimization is studied—local optimization corresponds to our optimal late service binding.

In this paper we address in a general setting the issues discussed in our running example. By encompassing AES (or contextual nets) with arbitrary confusion, we handle general orchestrations—we, however, require that executions are all finite; if not so, evaluating the end-to-end QoS makes no sense. We propose an algebraic framework for QoS generalizing the calculus developed in our running example. We explain how the end-to-end QoS of an orchestration can be defined and computed. We study monotonicity by providing conditions ensuring it and counter-measures to overcome the lack of it.

We use AES for our mathematical developments. The notion of AES was introduced in the companion paper [1] and we recall the corresponding notations in Figure 2. Our running example was presented using contextual nets. The companion paper [1] recalls how every contextual net defines an associated AES.

\[ E \] shorthand for AES \((E, \leq, \triangleright)\) where \(\triangleright\) denotes the asymmetric conflict
\[ [e] = \{x \in E \mid x \leq e\}, \text{ with } e \text{ an event of AES } E \]
\[ \text{Conf}(E) \] the set of configurations of AES \(E\)
\[ \text{Conf}(E) \] the set of maximal configurations of AES \(E\)
\[ \text{CC-Conf}(E) \] the set of CC-configurations of AES \(E\)
\[ \alpha, \beta \] notations for branching cells
\[ C \vdash_E \alpha \] configuration \(C\) of \(E\) enables the branching cell \(\alpha\)

Figure 2: Notations for Asymmetric Event Structures and Branching Cells

2. QoS Domains

In this section we formalize QoS Domains. The reader is referred to our running example of Figure 1, where motivations for the forthcoming definitions were given:

**Definition 2.1.** A QoS domain is a tuple \(Q = (Q, \preceq, \oplus)\) where:

- \((Q, \preceq)\) is a complete upper semi lattice; in other words, \((Q, \preceq)\) is a partially ordered set such that every subset \(S \subseteq Q\) has a least upper bound denoted by \(\bigvee S\). For any \(S \subseteq Q\), we respectively denote by \(\min(S)\) and by \(\max(S)\) the set of minimal, respectively of maximal elements of \(S\).

- \((Q, \oplus)\) is a commutative semi-group with neutral element 0 and such that:

\[(\text{monotonicity})\] \(\forall q, q' \in Q : q \preceq q' \iff q \oplus q' = q'\)

\[(\text{3})\] \(q \preceq q' \implies \exists q'' \in Q : q \preceq q' \oplus q''\)  \(\text{(4)}\)
Referring to our motivating discussion: $Q$ is the set in which QoS takes its values; $q \preceq q'$ is interpreted as “$q$ is better than (or preferred to) $q'$”; partial order $\preceq$ gives raise to the least upper bound $\lor$, interpreted as the worst QoS; operator $\oplus$ is used to accumulate QoS quanta from causally related events. Examples of QoS domains include:

- **Latency or Response Time**: the end-to-end QoS of a configuration gives the accumulated latency $d$, or “delay” of this configuration. Corresponding QoS domain is $Q = \mathbb{R}_+$, equipped with $\oplus_d = +$, and $\preceq_d$ is the usual order on $\mathbb{R}_+$. This is the basic example of QoS metric, which was studied in [6].

- **Security Level**: the QoS domain is $(Q_s, \preceq_s, \oplus_s)$, where $Q_s = \{\text{high}, \text{low}\}$ with $\text{high} \preceq_s \text{low}$, $\oplus_s = \lor_s$ reflecting that a low security service processing a high security data yields a low security response. More complex partially ordered security domains can be handled similarly.

We do not claim that this solves security in orchestrations. It only serves a more modest but nevertheless useful purpose, namely to propagate and combine security levels of the requested services to derive the security level of the orchestration. How security levels of the requested services is established is a separate and more difficult issue, addressed, e.g., by relying on reputation or through the negotiation of security contracts. Here we assume that services expose this information.

- **Reliability**: the QoS domain is as for Security Level, except that $\text{high}/\text{low}$ is replaced by nominal/faulty. By equipping this QoS domain with probability distributions (see the discussion in Section 6), we capture reliability in our setting.

**Running Example 6.** Running Example 2 uses multi-dimensional QoS, namely latency and security level, considered jointly. So, for our running example, $Q = Q_s \times Q_d$ equipped with the product order. □

Our model of a QoS domain is very close to the approach of Buscemi and Montanari (2011) [8], Buscemi and Montanari (2007) [7], De Nicola et al. (2005) [11]. In this set of works, generic QoS is supported through a commutative semiring algebra bearing tight similarities with $(Q, \preceq, \oplus)$—the only difference is the technical “archimedian” condition (4), which is used in the study of monotonicity. The cc-pi calculus [7] was proposed to model dynamic service binding with QoS-based selection and its expressiveness is studied.

### 3. Performance AES and On-line Optimal Processes

In this section we develop the framework of Performance AES to model composite service execution with QoS-based dynamic service binding.

#### 3.1. Performance AES and QoS of Configurations

The notion of Performance AES introduced below is a natural framework to relate causality processes with asymmetric conflict with QoS of service calls, captured by QoS domains introduced in the previous section.
Definition 3.1. A Performance AES (P-AES) is a tuple \( \mathcal{P} = (E,Q,\lambda_0,\lambda) \). \( E \) is a finite AES, \( Q = (Q,\preceq,\oplus) \) is a QoS domain, \( \lambda_0 \in Q \) is the initial QoS capturing the activation of the P-AES, and the total function \( \lambda : E \to Q \) is the QoS label, which associates, to each event, a QoS quantum.

Our first task is to relate the end-to-end QoS of the composite service to the QoS label \( \lambda \) attached to each individual service call. This is captured by the notion of end-to-end QoS of a configuration (see (1) and (2) of our running example), justified by the following theorem:

Theorem 3.2. Let \( E \) be an AES, \( Q \) a QoS domain and let \( \lambda : E \to Q \) be any given function. Then there exists a unique function \( \Lambda : \text{Conf}(E) \to Q \), called the end-to-end configuration QoS, such that, for any configuration \( C \) and any event \( e \):

\[
\begin{align*}
\Lambda(\emptyset) &= \lambda_0 \\
\Lambda(C) &= \bigvee_{e \in L(C)} \Lambda([e]) \quad \text{if} \quad C \neq \emptyset \\
\Lambda([e]) &= \lambda(e) \oplus \Lambda([e] \setminus \{e\})
\end{align*}
\]

where \( L(C) \) denotes the set of maximal events of \( C \) for the causality order \( \leq \) of \( E \) (the “top layer” of \( C \)). Moreover this function satisfies the following monotonicity property, for \( C, C' \) any two configurations:

\[
C \subseteq C' \implies \Lambda(C) \preceq \Lambda(C').
\]

We observe that \( \Lambda \) in Theorem 3.2 is given by two coupled definitions. We thus need to prove that such a definition makes sense and uniquely defines \( \Lambda \). The proof is found in Appendix A.1.

Thus, for \( \mathcal{P} = (E,Q,\lambda) \) a P-AES, Theorem 3.2 uniquely defines the end-to-end QoS of any configuration \( C \), we denote it by \( \Lambda_\mathcal{P}(C) \) in the sequel.

Running Example 7. Running Example 3 discusses, for the different executions that can occur in the orchestration of Figure 1, how the end-to-end QoS is computed according to formulas (5).

3.2. On-line Optimization of the QoS

For \( X \) any set and any map \( \varphi : X \to Q \), the notation

\[
(\text{min}, \arg\min) \{ \varphi(x) \mid x \in X \}
\]

denotes a Pareto minimum, i.e., any pair \((x, \varphi(x))\) such that no \( x' \in X \) exists with \( \varphi(x') \prec \varphi(x) \). Pareto minima or maxima are thus not uniquely defined.

QoS-based service selection is a typical operation in composite services. A straightforward policy is the a posteriori best service selection, which consists in finding:

\[
(Q_{\text{opt}}(\mathcal{P}), C_{\text{opt}}(\mathcal{P})) = (\text{min}, \arg\min) \left\{ \Lambda_\mathcal{P}(C) \mid C \in \text{Conf}(E) \right\}
\]

in the sense of (7). Policy (8) is, however, not appropriate in a dynamic world of services. On-line late service binding is preferred, which consists in on-line selecting, at run time, one service among the enabled ones, based on its QoS. We thus abandon policy (8) and replace it by an on-line selection policy with
no backtrack, in which a selection among competing services occurs each time
the due information is at hand for this selection to be sound.

Choosing among competing alternatives can only be performed at certain
stages of the execution, where the due information for consistently identifying
the possible alternative continuations is available. The **CC-configurations** in-
troduced in companion paper [1] identifies the subclass of configurations where
such a choice can be properly made. **Branching cells**, also introduced in [1],
identify the minimal sets of events involved in such a choice. A key result
of [1] states that any maximal configuration $C$ is covered by the branching cells
capturing the successive choices made when executing it: $C \subseteq \bigcup_i \alpha_i$; furthermore, its prefixes of the form $C_n = C \cap (\bigcup_{m \leq n} \alpha_m)$ are the **CC-configurations**
converging to $C$ (the convergence occurs in finitely many steps).

Thus, in our framework, choosing among competing alternatives consists in
extending a **CC-configuration** by means of a branching cell and then choosing a
maximal configuration of the latter. The procedure we develop next formalizes
this—we used it in our running example of Section 1. Recall that $L(C)$ denotes
the set of maximal events of $C$ for the causality order $\leq$ of $E$, i.e., the “top
layer” of $C$:

**Procedure 1** (execution with late service binding). Execute the following loop
with initial Data: $=$ $\langle \emptyset, \emptyset \rangle$ in order to maintain the set

$$\Lambda_P(C) = \{ \Lambda_P([e]) \mid e \in L(C) \} \text{ for } C \in \text{CC-Conf}(E),$$

from which the end-to-end QoS of $C$ can be computed by $\Lambda_P(C) = \bigvee \Lambda_P(C')$:

**Data**: $(C, \Lambda_P(C))$ with $C \in \text{CC-Conf}(E)$.

1. Select a branching cell $\alpha$ enabled at $C$ in $E$, i.e., $C \vdash_E \alpha$;
2. Using recursive formula (5), evaluate $\Lambda_P(C \uplus C')$ for $C'$ ranging
   over $\text{Conf}(\alpha)$, starting from $C' = \emptyset$, and select a Pareto
   minimum

   $$(\Lambda_P(C_*, C_*)) = (\min, \arg \min) \left\{ \Lambda_P(C \uplus C') \mid C' \in \overline{\text{Conf}(\alpha)} \right\}.$$

3. If $C \uplus C_*$ is a maximal configuration, then stop the loop; else
   compute $\Lambda_P(C \uplus C_*)$ and restart from Step 1 with
   **Data**: $=$ $\langle C \uplus C_*, \Lambda_P(C \uplus C_*) \rangle$.

The loop eventually stops when reaching some maximal configuration $C_{\max}$ and
we set

$$(Q_{\text{online}}(P), C_{\text{online}}(P)) = \{ \Lambda_P(C_{\max}), C_{\max} \}. \quad (9)$$

Procedure 1 with its formula (9) formalizes QoS-based late service binding
and evaluates the resulting end-to-end QoS for the composite service.

Non deterministic occurs in Procedure 1 in two distinct places at each round
of the loop: in Step 1, when selecting a branching cell among the several branching
cells enabled at $C$; and in Step 2, since the Pareto minimum is not unique.
The following result shows that the first kind of non deterministic at least has
little influence on the final result of the procedure. See a situation in § 4 and in
particular in Theorem 4.5 where both sources of non deterministic are uncorre-
lated.
Proposition 3.3. Let $C$ be a maximal configuration selected by Procedure 1. Then for any choice of successive branching cells $\alpha$ in Step 1 of the procedure, there is a corresponding choice in Step 2 of the procedure such that the resulting maximal configuration selected is $C$.

Proof. Let $\alpha$ be any branching cell selected by Step 1 in the first round of Procedure 1. Proceeding inductively, it suffices to show that there is some optimization choice in Step 2 that precisely picks $C_\alpha = \alpha \cap C$ as optimum. Since $\alpha$ is selected at Step 1, $\alpha$ is an initial branching cell of $E$. of $C$, and thus is selected at some round, say $n$, of Step 1 in the loop that produces $C$. By hypothesis, $C_\alpha = \alpha \cap C$ is selected as a Pareto minimum inside branching cell $\alpha$ in Step 2 of the same round $n$ of the loop. The local form (5) of the function $\Lambda$ then shows that the very same $C_\alpha$ is a valid candidate for the Pareto optimum in Step 2 of the first Round of the original loop, which proves the lemma.

Despite Proposition 3.3, non determinism still arises in formula (9), due to the non uniqueness of the Pareto minimum in step 2. Indeed, formula (9) non deterministically selects a candidate among a subset of QoS values. Remark that the a posteriori best selection service given by the optimization problem (8) is also non deterministic, for the same reason that minimal elements are not uniquely defined.

Running Example 8. The discussion of Running Example 3 illustrates how late service binding according to Procedure 1 is performed in our running example, when choosing between the two services $S_{11}$ and $S_{12}$.

3.3. Comparing On-line and Off-line Best Services

Since the a posteriori optimization problem (8) contrasts with the on-line selection service given by Procedure 1, we shall call it in the sequel off-line optimization problem. The comparison between off-line and on-line optimization services is a natural question.

To compare subsets of QoS values, for $X, X' \subseteq Q$, say that

\begin{align}
X &\prec X' \quad \text{if} \quad \forall q \in X, \exists q' \in X' \text{ such that } q \preceq q' \\
X &\preceq X' \quad \text{if} \quad \forall q' \in X', \exists q \in X \text{ such that } q \preceq q' 
\end{align}

(10) (Note the two distinct cases). Using notation (10) the following holds:

Proposition 3.4. For any P-AES $\mathcal{P}$, $Q_{\text{opt}}(\mathcal{P}) \prec Q_{\text{online}}(\mathcal{P})$ holds.

Proof. By definition of $Q_{\text{opt}}(\mathcal{P})$, we have $Q_{\text{opt}}(\mathcal{P}) \subseteq \{ \Lambda_{\mathcal{P}}(C) \mid C \in \text{Conf}(E) \}$. The result follows by observing that $Q_{\text{online}}(\mathcal{P}) \subseteq \{ \Lambda_{\mathcal{P}}(C) \mid C \in \text{Conf}(E) \}$. 

Finally, the a posteriori worst case end-to-end QoS will also be useful considering (we are not interested in the argument of the maximum):

\begin{align}
Q_{\text{worst}}(\mathcal{P}) &= \max \left\{ \Lambda_{\mathcal{P}}(C) \mid C \in \text{Conf}(E) \right\} .
\end{align}

(11)
4. The Issue of Monotonicity

A basic assumption underpinning the management of composite services is that QoS improvements in the called services can only be better for the composite service. We refer to this as the *monotonicity* property. Unfortunately, monotonicity is by no means a trivial assumption when late service binding is considered. Observe that Procedure 1 differs from Belmann dynamic programming [5]: the latter is a backward incremental computation of (8), whereas Procedure 1 proceeds forward by short horizon decisions, which may impair monotonicity. In this section we study conditions by which monotonicity still holds under on-line service selection in Procedure 1, we call this property on-line monotonicity. This notion was first noticed in [16, 6]. The issue of monotonicity was discussed in Running Example 4.

To formalize this concept we need some further notations. Fix an AES $E$ and a QoS domain $Q$. For $\lambda, \lambda' : E \to Q$ two QoS labels, write $\lambda \preceq \lambda'$ if and only if, for every $e \in E$, $\lambda(e) \preceq \lambda'(e)$ holds. Monotonicity will depend on both structural conditions on the underlying AES and on the set $L$ of allowed QoS labels $\lambda : E \to Q$.

**Definition 4.1** (on-line monotonicity). Let $(E, Q, L)$ be a triple consisting of an AES $E$, a QoS domain $Q$, and a set $L$ of allowed QoS labels. Using the notation introduced in (10), say that $(E, Q)$ is on-line monotonic if, for any two $P$-AES $P = (E, Q, \lambda_0, \lambda)$ and $P' = (E, Q, \lambda'_0, \lambda')$ such that $\lambda, \lambda' \in L$,

$$\lambda_0 \preceq \lambda'_0 \quad \lambda \preceq \lambda' \quad \Rightarrow \quad Q_{\text{online}}(P) \preceq Q_{\text{online}}(P')$$  \hfill (12)

**Definition 4.1** expresses that improving the QoS labels of each event improves the end-to-end QoS of the P-AES. The following result follows directly from the properties of QoS domains:

**Lemma 4.2.** Using the notations introduced in Definition 4.1, if $\lambda_0 \preceq \lambda'_0$ and $\lambda \preceq \lambda'$ then:

$$Q_{\opt}(P) \preceq Q_{\opt}(P') \quad Q_{\worst}(P) \preceq Q_{\worst}(P')$$  \hfill (13)

**Proof.** By definition of $Q_{\opt}(P)$, we have $Q_{\opt}(P) \preceq \{ \Lambda_{P}(C) \mid C \in \overline{\text{Conf}(E)} \}$. By monotonicity of $\lambda \mapsto \Lambda_{P}(C)$ for any fixed maximal configuration $C$ and $P$ associated to $\lambda$, we get $\{ \Lambda_{P}(C) \mid C \in \overline{\text{Conf}(E)} \} \preceq \{ \Lambda_{P'}(C) \mid C \in \text{Conf}(E) \}$. On the other hand, $\{ \Lambda_{P}(C) \mid C \in \text{Conf}(E) \} \preceq Q_{\opt}(P')$. Chaining these three $\preceq$-inequalities yields the second inequality of (13). Similarly, by definition of $Q_{\worst}(P')$, we have $\{ \Lambda_{P'}(C) \mid C \in \overline{\text{Conf}(E)} \} \succeq Q_{\worst}(P')$, and the rest of the former argument can be repeated, mutatis mutandis.

**Theorem 4.3.** $(E, Q, L)$ is on-line monotonic if, for any initial QoS $\lambda_0$ and any QoS label $\lambda \in L$, applying Procedure 1 to the P-AES $P$ defined by $\lambda$, always returns a Pareto optimum for (8).
Proof. A direct consequence of Lemma 4.2. □

The forthcoming Theorem 4.5, has the advantage of stating a structural condition on the AES involved. It is convenient to introduce first a definition:

**Definition 4.4.** Let $Q$ be a QoS domain, and let $\lambda : E \rightarrow Q$ and $\lambda' : E' \rightarrow Q$ be two mappings where $E$ and $E'$ are two AES. A mapping $\psi : E \rightarrow E'$ is an isomorphism of labeled AES between $(E, \lambda)$ and $(E', \lambda')$ whenever:

1. $\psi$ is a bijection, and for all $e_1, e_2 \in E$, one has $e_1 \leq e_2 \iff \psi(e_1) \leq \psi(e_2)$ and $e_1 \nearrow e_2 \iff \psi(e_1) \nearrow \psi(e_2)$; and
2. $\lambda' \circ \psi = \lambda$.

We say that $(E, \lambda)$ and $(E', \lambda')$ are isomorphic labeled AES if there exists such an isomorphism $\psi : E \rightarrow E'$.

**Theorem 4.5.** The following condition is sufficient for $(E, Q, L)$ to satisfy the condition of Theorem 4.3, and hence to be on-line monotonic:

For any labeling function $\lambda \in L$, for any $C \in \text{CC-Conf}(E)$, for any branching cell $\alpha$ enabled by $C$, and for any two maximal configurations $W, W' \in \text{Conf}(\alpha)$, the futures $E^{\text{Conf}W}$ and $E^{\text{Conf}W'}$ equipped with the restrictions of $\lambda$ to $E^{\text{Conf}W}$ and to $E^{\text{Conf}W'}$ respectively, are two isomorphic labeled AES.

(14)

The proof is found in Appendix A.2.

**Remark 1.** Structural condition (14) is very natural if AES $E$ is the unfolding of a safe contextual Petri net$^1$ In this case, every branching cell is folded as a subset of some cluster—recall that clusters are minimal subsets of transitions $t$ and places $p$ of the net satisfying

\[
\begin{align*}
\forall t \in c & \implies \quad t^* \subseteq c \\
\forall p \in c & \implies \quad p^* \subseteq c \\
\forall p \in c & \implies \quad \{ t \in T \mid \exists t \ni p \} \subseteq c
\end{align*}
\]

(15)

where $n^*$ and $n^*$ are the preset and postset of node $n$ and $t^*$ is the context of transition $t$, see [4]. The structural condition is in particular satisfied if all clusters $c$ of this net are such that, for any two transitions $t_1 \neq t_2$ belonging to $c$, $t_1^* = t_2^*$ holds. For this case of contextual net unfoldings, the condition $\lambda \circ \psi = \lambda$ from Definition 4.4 expresses that QoS labels are attached to transitions, a very natural condition.

It is shown in [6] that, for the particular case where the QoS is the response time (and under a mild condition on $L$), the conditions of Theorem 4.5 are also necessary.

---

$^1$It is more natural to consider the case of nets when discussing condition (14). The reader is referred to Appendix B of Companion paper [1] for an account on the unfolding of contextual nets.
5. Enforcing Monotonicity

Theorem 4.5 provides guidelines regarding how to enforce monotonicity. We illustrate this using our running example of Section 1.

One way of enforcing monotonicity is by invoking Theorem 4.5. Aggregate the two successive transitions \(a_{12}\) and \(b_{12}\), which capture the call and response of service \(S_{12}\), respectively. The resulting P-AES satisfies the condition of Theorem 4.5 and thus is on-line monotonic. This was the counter-measure proposed in Running Example 5.

An alternative to the above procedure consists in not modifying the orchestration but rather changing the QoS evaluation procedure. Referring again to Figure 1, isolate the part of the workflow that is a source of non-monotonicity, namely the subnet collecting the events \(a_{11}, a_{12}, b_{12}\). For this subnet, use pessimistic formula (11) to get a pessimistic but monotonic bound for the QoS of this subnet. For this example, the pessimistic bound is equal to

\[
\max(d_{11}, d_{12} + d'_{12}), \max(\text{high}, \text{low})
\]

We then plug the result in the evaluation of the QoS of the overall orchestration, by aggregating the isolated subnet into a single transition \(a_{12}b\), with this QoS increment.

The above two procedures yield different results. By aggregating service calls performed in sequence, the first procedure delays the selection of the best branch. The second procedure does not suffer from this drawback. In turn, it results in a pessimistic evaluation of the end-to-end QoS. Both approaches restore monotonicity.

6. Conclusion

By allowing service calls to be guarded by contexts, Asymmetric Event Structures and contextual nets are a convenient framework to model composite Web services or service orchestrations. We have proposed to equip AES with QoS domains as a framework to capture a number of QoS metrics and their combination. We used the resulting model to formalize QoS-based late service binding in composite services. Branching cells of AES played a central role in this formalization by localizing choices in a precise way. When subject to QoS-based late service binding, composite services may be non-monotonic with respect to QoS, meaning that strictly improving the QoS of a service may strictly decrease the end-to-end QoS of the composite service, an embarrassing feature for QoS-aware management; we provided sufficient conditions ensuring monotonicity and guidelines for enforcing it.

Now, the QoS offered by services is typically varying and subject to uncertainty. Our framework is prepared for a probabilistic extension dealing with uncertainty, see [16] for an informal presentation of this. By identifying all sources of nondeterminism and then randomizing them, all results developed here extend to the probabilistic setting. This probabilistic extension is essential to capture reliability as proposed in Section 2.

This theoretical investigation paves the way to provide formal support for the separation of concerns in the management of composite services. It would be desirable to see function and QoS as two cross-cutting “aspects” of a composite service. Each aspect should be developed separately using a “minimal”
specification of the interaction with the other aspect. Weaving these two aspects should return the program specifying the QoS-enhanced composite service. Our generic model of QoS domain should get instantiated for each QoS metric of interest, using a library of predefined QoS classes. A subset of the authors has started the development of a framework supporting all of this on top of the Orc language developed by the team of J. Misra [13], this development will be reported elsewhere.

A. Missing proofs

A.1. Proof of Theorem 3.2
The proof is by induction on the height of a configuration, using the following explicit construction:

1. For $C \in \text{Conf}(E)$, we define the height $\tau(C) \in \mathbb{N}$ of $C$ as the longest length of a causality chain in $C$.
2. For $n \geq 0$ we set $C_n = \{ C \in \text{Conf}(E) \mid \tau(C) \leq n \}$.
3. We define by induction the sequence $(\Lambda_n)_{n \geq 0}$ of mappings $\Lambda_n : C_n \rightarrow Q$:
\[ \forall n \geq 0 : \Lambda_n(\emptyset) = \lambda_0 \]
\[ \forall n \geq 0, \forall C \in C_{n+1} : \Lambda_{n+1}(C) = \bigvee_{e \in L(C)} \left( \Lambda(e) \oplus \Lambda_n([e] \setminus \{e\}) \right) \]
4. Finally we put, for $C \in \text{Conf}(E)$:
\[ \Lambda(C) = \Lambda_{\tau(C)}(C). \] (A.1)

Lemma A.1. Let $C \in \text{Conf}(E)$ and $n = \tau(C)$. Then $\Lambda_p(C) = \Lambda_n(C)$ for all integer $p \geq n$. Thus, we have, for any configuration $C$ and any integer $n$:
\[ n \geq \tau(C) \implies \Lambda(C) = \Lambda_n(C). \] (A.2)

Proof. It suffices to prove by induction on integer $n \geq 0$ the following property:
\[ (H_n) : \forall C \in \text{Conf}(E) \forall p \in \mathbb{N} \tau(C) = n \land p \geq n \implies \Lambda_p(C) = \Lambda_n(C). \]
The property is obvious for $n = 0$. Assuming $(H_n)$ holds, let $C \in \text{Conf}(E)$ and $p$ be an integer such that $\tau(C) = n + 1$ and $p \geq n + 1$. Then for any $e \in L(C)$, we have that $\tau([e] \setminus \{e\}) \leq n$, and since $p - 1 \leq n$, it follows from $(H_n)$ that $\Lambda_{p-1}(e) = \Lambda_n(e)$. By definition of $\Lambda_p$ and of $\Lambda_{n+1}$, we calculate thus as follows:
\[ \Lambda_p(C) = \bigvee_{e \in L(C)} \left( \Lambda(e) \oplus \Lambda_{p-1}([e] \setminus \{e\}) \right) \]
\[ = \bigvee_{e \in L(C)} \left( \Lambda(e) \oplus \Lambda_n([e] \setminus \{e\}) \right) = \Lambda_{n+1}(C), \]
completing the proof of the induction.

Lemma A.2. The function $\Lambda$ defined above by $\Lambda(C) = \Lambda_{\tau(C)}(C)$ satisfies formulas (5).

Proof. Easy using (A.2) and the inductive definition of $(\Lambda_n)_{n \geq 0}$.
Lemma A.3. The function $\Lambda$ satisfies, for any two configurations $C$ and $C'$:

$$C \subseteq C' \implies \Lambda(C) \preceq \Lambda(C').$$  \hfill (A.3)

Proof. By induction over $\tau(C)$. \hfill $\square$

Proof of Theorem 3.2. The existence follows from Lemmas A.2 and A.3. For the uniqueness, prove by induction over $\tau(C)$ that, for any $\Lambda$ satisfying (5), then $\Lambda(C) = \Lambda_n(C)$ for all $C \in \text{Conf}(E)$ of height $\leq n$. \hfill $\square$

A.2. Proof of Theorem 4.5

Proof of Theorem 4.5. We first observe that, if $C$ and $C'$ are two maximal configurations of $E$, their coverings by branching cells given by Theorem 3.7 of companion paper [1] have the same cardinality, and can be ordered respectively as $(\alpha_1, \ldots, \alpha_n)$ and $(\beta_1, \ldots, \beta_n)$, in such a way that, for all $i = 1, \ldots, n$, when $\alpha_i$ and $\beta_i$ are equipped respectively with the restrictions $\lambda|_{\alpha_i}$ and $\lambda|_{\beta_i}$, then $(\alpha_i, \lambda|_{\alpha_i})$ and $(\beta_i, \lambda|_{\beta_i})$ are isomorphic AES.

The proof is thus by induction on the length $n$ of the above coverings:

Induction hypothesis: Theorem 4.5 holds for a covering length $\leq n - 1$.

Let $C = W_1^C \uplus W_2^C \uplus \ldots \uplus W_n^C$ be any maximal configuration of $E$ together with its decomposition according to Theorem 3.7 of companion paper [1], and let $(\alpha_1^C, \ldots, \alpha_n^C)$ be the corresponding covering by branching cells. Recalling that $L(W_i^C)$ denotes the set of maximal events of $W_i$ for the causality order $\preceq$ of $E$, restrict $E$ to $E_{1,C} = \text{def} \ L(W_i^C)|_{E_i^C}$ and then equip $E_{1,C}$ with $\lambda_1 = 0$ and $\lambda^1(e) = \lambda(e)$, except for $e \in L(W_i^C)$ for which we set $\lambda^1(e) = \Lambda_P(\lfloor e \rfloor)$. Denote by $P^1$ the resulting P-AES.

Lemma A.4. The following holds, for $C \in \overline{\text{Conf}(E)}$:

1. By (14), the P-AES $(E_{1,C}, \lambda|_{E_{1,C}})$ are all isomorphic;
2. By Theorem 3.2, $\Lambda_{P^1}(C|_{E_{1,C}}) = \Lambda_P(C)$;
3. The induction hypothesis holds for the futures $(E_{1,C}, Q, L|_{E_{1,C}})$.

Applying Lemma A.4 for $C = C_{\text{max}}$ a maximal configuration constructed by Procedure 1, we complete the proof by contradiction. Assume that $C_{\text{max}}$ is not Pareto optimal for (8). Then, there exists a maximal configuration $C$ of $E$ such that $\Lambda_P(C) \prec \Lambda_P(C_{\text{max}})$. Hence, at least one of the following two cases must occur:

1. $\Lambda_P(C|_{\alpha_i^C}) \prec \Lambda_P(C_{\text{max}}|_{\alpha_i^C_{\text{max}}})$, which would contradict step 2 of Procedure 1;
2. Hence, by property 2 of Lemma A.4, $\Lambda_P(C|_{E_{1,C}^C}) \prec \Lambda_P(C_{\text{max}}|_{E_{1,C}^C_{\text{max}}})$ must hold, which would contradict properties 1 and 3 of Lemma A.4.

This completes the proof of Theorem 4.5. \hfill $\square$
References


