On Graph Reduction for QoS Prediction of Very Large Web Service Compositions

Alfredo Goldman and Yanik Ngoko
Institute of Mathematic and Statistics
University of São Paulo
São Paulo, Brasil
{ *gold,yanik* }*@ime.usp.br*

***Abstract*—In this paper, we investigate the question of QoS prediction of Web Service Composition (WSC) implementing a business process. We focus on the graph reduction technique and the prediction of the Service Response Time. In the graph reduction technique, we assume that a Web Service Composition can be represented as a graph. The main thesis is that the QoS of such a graph can be obtained from a composition of the ones of its nodes.**

Multiple graph reduction algorithms have been proposed in the literature. Our contribution is twofold. We first propose a fast algorithm based on graph reduction for the prediction of the Service Response Time of a Web Service Composition. In comparison to those existing in the literature, this algorithm uses less memory space and has a better time complexity. The obtained improvements are in particular significant on very large Web Service Composition where the number of services is huge. Our second contribution is an analysis of the graph reduction technique for QoS prediction that takes into account the unfolding of services. In such cases, we show that the prediction of QoS can lead to a NP-complete problem. We also provide an integer programming model for predicting the Service Response Time in this case.

***Keywords*-Web Service Composition; Graph reduction; Business process; QoS prediction; BPMN.**

I. Introduction

In this work, we are interested in the prediction of the Quality of Service of Web Services Compositions (WSC). We consider a setting comprising a finite set of Web Services executed on a finite set of Servers. Each Web Service has a set of operations that may participate in various Business processes [1]. At a moment, a user request is initiated in a business process; Its execution can be described as a subgraph that traverses multiple Web Services according to the interconnection structure of the business process. Our goal is to estimate the Service Response Time [2].

The QoS prediction on WSC has been investigated in multiple works. Many of them are based on the graph reduction technique [2], [3]. Three main stages are important for defining such a technique: the definition of a finite set of generic subgraph patterns, the definition of aggregations rules stating for each subgraph pattern how to infer the QoS values from the ones of its nodes, the definition of a reduction algorithm that proceeds by decomposing an arbitrary subgraphs in generic patterns and aggregating the QoS.

Several subgraphs structures, aggregations rules [2], [4], [3], [5] and graphs reductions algorithms [2], [6] exist for QoS prediction of WSCs. These works examined the prediction on various QoS dimensions as the Service Response Time, the throughput, the price, the reliability, the fidelity etc.

The graph reduction is a simple and modular technique for QoS prediction. One of its limitations is the hardness to handle unstructured compositions of Web Services [6]. Alternative techniques exist for QoS prediction. Most of them adopt a different representation of the service composition. We have for instance, Stochastic Automata Network representations and, workflow nets ones [7], [8].

In this work, we are interested in predicting the Service Response Time of WSCs with the graph reduction technique. Despite our restriction to Service Response Time, our study can easily be generalized to others QoS parameters following precursor works on graph reduction [2], [3].

We first propose a fast algorithm for graph reduction. In comparison to what exists in the literature [2], [6], this graph reduction algorithm uses less memory and perform fewer operations. The obtained improvements are significant when dealing with huge compositions of services. Our second contribution is an analysis of the graph reduction technique that takes into account the services unfolding. In such cases, we show that the prediction of QoS can lead to a NP-complete problem that we propose to solve with an integer programming based model.

The remainder of the paper is organized as follows: In the sections II and III, we discuss about Web Service Composition modeling. The sections V and IV deal with the Service Response Time prediction. We conclude in Section VI.

II. A model for Web Service Composition

In this section, we present our viewpoint about the structure of a WSC.

A. Web Service Composition

Our vision of a WSC is illustrated in Figure 1. The services composition is a Hierarchical Service Graph (HSG) with three layers: an operation and business process layer, a Web Service layer and a machine layer.

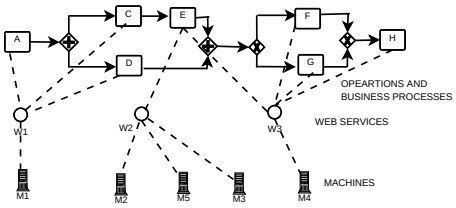


Figure 1: Example of Web Service Composition.

The machine servers and their interconnection constitute the machine layer of the HSG. At least one Web Service is unfolded on each machine. For instance, on the machine $M5$ of Figure 1, we have the Web Service $W2$. Each Web Service exists in one or many instances on a machine and, must implement at least one operation. For instance, $W1$ implements the operations A, C, D. An operation denotes an atomic action that can be realized from the public interface of a Web Service. In practice, user authentication or user account creation are typical operations supported by Web Services. A same operation can be implemented by more than one Web Service. Finally, the operations participate in a business process that we will consider as being an orchestration, as defined in the Business Process Modeling Notations (BPMN) [1]. This process in our work is given by an *operation graph* $G_o = (V \cup C, E_{vc})$. Here, V is the set of operations, C are connectors between operations and E_{vc} describes interconnection between operations and connectors.

This vision of WSC takes implicitly into account some common critical questions while designing WSC. We refer here to the selection of the adequate Web Service for executing an operation or the planning of Web Services on machines in order to optimize the QoS [6], [9]. Our focus, we recall, is on the QoS prediction.

As defined above, a HSG admits the possibility of having a same Web Service, unfolded on multiple machines. More, multiple Web Services can implement the same operations. In the sequel however, we will assume that an operation is implemented only by a single Web Service that itself can be unfolded only on a single machine. For qualifying this restriction, we will say that such HSGs are *injective between the layers*.

In the rest of the paper, HSG will be considered as the basic structure for defining a WSC. Our main focus on such graphs will be on the operations level where a BPMN orchestration is described. We will not take into account all available BPMN constructs for orchestration design. The next section defines the restricted set of BPMN elements on which we will focus.

B. BPMN patterns for the operations level

We will restrict the set of possible business processes to a finite set of patterns frequently used in business process modeling. The chosen patterns here are: sequence, fork, inclusive choice and exclusive choice. An illustrative description of these patterns in the BPMN notations is given in Figure 2. We provide a textual description in the sequel.

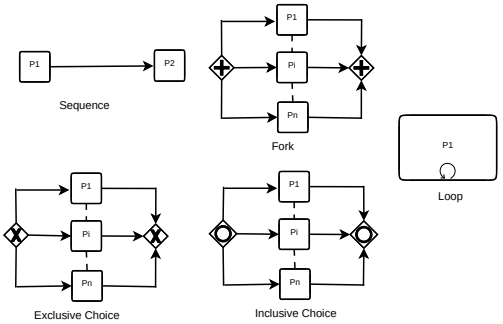


Figure 2: Pattern in process definition

1) Sequence and loop pattern: In the sequence pattern, we have two subgraphs P_1, P_2 of $G_o = (V \cup C, E_{vc})$ that are connected. Each subgraph can be an operation, or, a combination of subgraphs under the given pattern structures. The loop pattern is given by a single operation that can be executed repetitively. In practice, it might happen that a loop comprises a subgraph instead of a single operation. Even if we do not perform this case here, our solutions can easily be generalized to handle it.

2) Fork, exclusive and inclusive pattern: In the fork pattern, we have two opening and closing *Fork connectors* that are linked to n subgraphs P_i. These latter ones may be reduced to operations or subgraphs combinations. The inclusive and exclusive patterns respect the same rules. The only difference are the used connectors (exclusive and inclusive connectors).

One can notice that we restricted ourself in this work to structured BPMN patterns. This choice is, amongst other things, justified by the existence of algorithms for turning arbitrary BPMN patterns into structured ones [10].

In the patterns illustration of Figure 2, we considered P_i as subgraphs. In the special case, where each P_i is an operation, we will say that we have an **elementary BPMN pattern**. In the next section, we introduce other concepts for the manipulation of HSGs.

3) Other concepts on Hierarchical Service Graphs: We will say that an operation graph $G_o = (V \cup C, E_{vc})$ is **decomposable** if its interconnection structure can be described as a recursive composition of sequence, fork, exclusive and inclusive patterns. For instance, a sequence operation A connected to a decomposable subgraph P_1, is, a decomposable graph.

On decomposable graphs, we make the following observation:

***Fact** 1:* If an operation graph $G_o = (V \cup C, E_{vc})$ is decomposable, then, each operation and opening connector

has at most one predecessor; There is also a unique node $u \in V \cup C$ without predecessors and a unique node without successors in G_o.
This fact follows from the description of the BPMN patterns; It will have a considerable importance in this work. The unique node without predecessors will be called **root node** and the unique one without successors **leaf node**.

For capturing the possible encapsulations of subgraphs in an operation graph, we propose to associate to all opening connectors a **deepness** parameter defined in the sequel.

***Definition** 1:* Given an operation graph G_o, let us suppose that for an opening connector u, we have n paths $Pt_1, \dots Pt_n$ leading to it. In each path Pt_i, we have α_i opening connectors and β_i closing connectors. The deepness of u is defined as $\max_{1 \leq i \leq n} \{\alpha_i - \beta_i\}$
From the fact 1, we can deduce that on a decomposable graph, there is always an opening connector with a null deepness.

We presented the basic structure that we will use for studying WSC. Our goal is to estimate the Service Response Time (SRT) of Web Service Compositions. In the next section, we deal with the modeling of the SRT on Hierarchical Service Graph.

III. Modeling the Service Response Time

A. Execution of a user request on Hierarchical Service Graph

For each user request, we denote by its execution trace, the subgraph of operations connectors and links that the *request traverses*. This subgraph can be reduced to a path between operations and connectors, if, it does not have a BPMN inclusive connector. We will assume in this work that given the execution trace of a request, the SRT is concentrated on the operations that it contains. For each operation u, we also assume that we have a parameter t_u, that is, the value of its mean response Time. This time can be obtained by running many times the operation and getting the mean resulting time. Our main thesis in SRT modeling is that the SRT of an operation graph can be obtained by composing the ones of its operations. We will see how this composition can be done in the next section.

B. SRT modeling on subgraphs

We only consider here operation graphs composed of subgraphs that can be described with elementary BPMN patterns (see Section II-B). We qualify such subgraphs as *elementary*. In table I, we propose some aggregations rules for computing the SRT of elementary subgraphs knowing the ones of their operations. Our rules are essentially based on the proposed in [2]. In the aggregations, we assumed that an operation can start its execution as soon as all required inputs are available. In practice, however, other overheads (due for example to queuing phenomenon on network) must be taken into account.

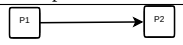
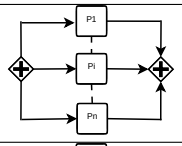
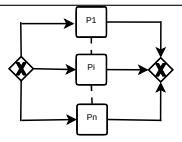
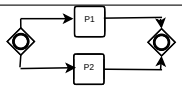
Elementary subgraph	Representation	SRT
Sequence		$t_{p_1} + t_{p_2}$
Loop		$\sum_{i=1}^{n_1} pl_i.i.t_{p_1}$
Fork		$\max\{t_{p_1}, \dots, t_{p_n}\}$
Exclusive		$\sum_{i=1}^{n} pr_i.t_{p_i}$
Inclusive		$ps_1.t_{p_1} + ps_2.t_{p_2} + ps_3.\max\{t_{p_1}, t_{p_2}\}$

Table I: Service Response Time on elementary subgraphs

For a loop operation P_1, we assumed that we know the maximal number of iterations n_1 that it can comprises. We also assume that we have the probability pl_i of executing the loop i times.

For the fork subgraph, we only presented the aggregation in one case. Indeed, it can happen for example that P_1 and P_n are operations of the same Web Service in a fork subgraph. This is the case for example on Figure 1 with the operations C and D. If the Web Service only exist in one instance, then the concurrent execution of these operations will not be possible. In such situations, we will say that the business process execution is serialized. If, however, all operations can use a different Web Service instance, then we can execute them in parallel. The SRT presented in the table are for this latter case. The case where the execution can be serialized will be considered in Section V.

For the sake of simplicity, we assumed in this work that open inclusive connector are always connected to only two paths (they have an outgoing degree equal to 2). The probability ps_1 and ps_2 are the one that we have for a request to take the first or the second path after *traversing this connector* and ps_3 is the one for taking the two paths. In simplifying the structure of the inclusive connector, we reduce the potential combinatory explosion required when the outgoing degree of such connector is arbitrary.
Finally, on exclusive connectors, we assumed that we have the probability pr_i for a request to take the path P_i.

One might criticize our SRT modeling because of the restricted set of patterns and the assumption of deterministic execution times. About the set of patterns, let us underline that more elaborated BPMN patterns and aggregations rules can be found in [2], [4], [3], [5]. However, our pattern set is sufficient for defining a new graph reduction approach that can be extended on other BPMN patterns. About the deterministic setting, we remark that an alternative modeling

dealing with probabilistic execution time can be found in [5]. Despite its interest, we point out that in such cases, the modeling requires more input information. Moreover, the SRT estimation on the entire composition is then exponential on the number of Web Services. Thus, it cannot then handle large composition of services. In the next section, we consider the question of QoS prediction admitting that for all fork subgraph, there is no serialization constraint.

IV. Estimation of the Service Response Time without serialization constraints

We are interested in the prediction of the mean execution time for a user request when we do not have a serialization constraint. The absence of serialization constraint means that we can always execute parallel paths simultaneously as defined in the business process specification. We point out that most works on QoS prediction are done under this assumption.

A. Graph reduction

Given an operation graph $G_o = (V \cup C, E_{vc})$, the graph reduction technique proceeds by multiple steps where a subgraph of G_o is substituted by a single node. At the step i of the reduction, a decomposable subgraph G_i is chosen. The SRT of this subgraph is computed; The subgraph is then replaced by a new created operation whose SRT is set as the one of G_i (this replacement is also called the reduction of G_i). One proceeds like this until G_o contains a single operation. The SRT of this node is then returned as the SRT of G_o. Figure 3 gives an illustration of this technique.

For easing graph reduction, a first step of the graph reduction can consist in replacing all loop operations by non-looping ones. Thus, we will have a graph without the loop pattern. We will admit in the remainder that for the graph to reduce, all loop operations have already been replaced. It

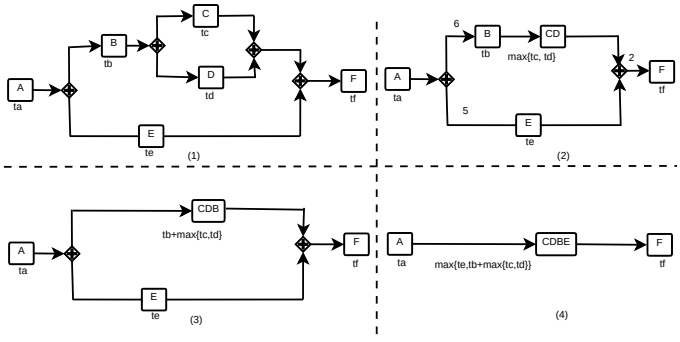


Figure 3: Example of graph reduction

might happen in the reduction that we want to substitute a *fork* subgraph whose branches contain a fork, exclusive or inclusive subgraph. On Figure 3, this is the case if we want to reduce the fork subgraph composed of operations nodes B, C, D, E at the beginning. For reducing this subgraph we need first to reduce all the inner fork, exclusive or inclusive subgraphs. This challenge has led to, at least, two types of reduction approaches in the literature. We discuss them in the sequel.

In [2], one of the most popular graph reduction algorithm (called SWR) was proposed. SWR proceeds by arbitrarily choosing the subgraph to reduce. When face to a subgraph containing an inner fork, exclusive or inclusive connector, the algorithm will look for another one. The idea is that in repeating this process, one will certainly reduce for all subgraphs their inner connectors. SWR algorithm admits redundant attempts of reduction of the same subgraph. On large graphs, this certainly has an impact on the execution time. A recursive approach that possibly can avoid redundant attempts of reduction has been proposed in [6]. However, as the approach is recursive, we have in the execution additional overhead on time and memory utilization caused, for example, by the management of the stacks of functions. More important, this approach deals with unstructured BPMN patterns while we focus on structured ones. In the sequel, we will propose an iterative approach for graph reduction that avoids redundant attempts of reduction on structured BPMN patterns.

B. A two phases approach for graph reduction

Our graph reduction approach proceeds in two phases. The first phase deals with the computation of a reduction order. The idea in such an order is to define an ordered set of decomposable subgraphs for the reduction. The second stage is the reduction with the defined order. We discuss these stages below.

C. Elementary reduction order

We define an elementary reduction order as follows:

***Definition** 2:* Given a graph G_o, a reduction order is an ordered list of decomposable subgraphs $\langle G_1, G_2, \ldots, G_n \rangle$ of G_o such that: 1) G_1 is an elementary subgraph; 2) After the reduction of $G_1, \ldots, G_i$, the subgraph G_{i+1} is elementary; 3) After the reduction of $G_1, \ldots, G_n$, G_o has only one node.

An important question with the definition above consists in knowing whether or not given a decomposable graph, such a reduction order is always possible. We establish that it is the case in the following.

***Lemma** 1:* If an operation graph G_o is decomposable, then, G_o contains at least one elementary subgraph.

Proof: For the proof, we use the deepness concept defined in Section II-B3. We will consider two cases. Given an operation graph G_o, let us suppose that it does not have an opening connector. Then, we only have sequence pattern with operations. In this case, We can easily find an elementary sequence subgraph in G_o. Let us now suppose that G_o has at least one opening connector. In this case, let us consider any opening connector with maximal deepness. Since the connector is of maximal deepness, on each of its

branches, we do not have an opening connector. Thus, either we have a sequence graph (but only with operations) on a branch, or we have only one operation on all branches. If we have a sequence with operations on a branch, then we have an elementary graph. In the case where we have only a single operation on all its branches, we have again an elementary where this opening connector is the root node. ■

***Lemma** 2:* If an operation graph G_o is decomposable, then, after the reduction of an elementary subgraph of G_o, we have again an elementary subgraph in G_o or G_o is reduced to an operation.

Proof: After the reduction of an elementary subgraph in G_o, we have two cases. If G_o is an operation, then we have the proof. If it is not the case, then, we can easily notice that G_o will be an operation graph that is decomposable. From the lemma 1, we can deduce that it will have an elementary subgraph. ■

Lemma 2 can also explain why the SWR algorithm does not loop infinitely. Indeed, it states that in considering all subgraphs, we will always find a reducible one.

***Theorem** 1 (**Existence of an elementary reduction order**):* If an operation graph G_o is decomposable, then, there exists at least one elementary reduction order

This theorem is a direct consequence of the previous lemmas. In the next section, we will propose an algorithm for computing a reduction order.

D. An algorithm for computing elementary reduction order

For representing a reduction order, we first notice that from the fact 1, a couple (r, l) where r is a root node and l a leaf one is sufficient to represent a decomposable subgraph. Thus, we can store a reduction order $\langle G_1, \ldots, G_n \rangle$, as n couples $(root, leaf)$. The benefit is that we will need a memory space in $O(n)$ memory space instead of the $O(n.|V \cup C|))$ required if we represented all subgraphs. In addition, to improve the memory storage, this choice as we will see is advantageous for simplifying the design of our algorithm.

We assume in the algorithm that for any operation u, $next(u)$ defines its direct successor. We also use the classical primitives $Push(S, e)$ and $Pop(S, e)$ for putting and retrieving an element e at the top of a stack S and, a special primitive $Push_x(S, e)$ that proceeds as $Push$ at the difference that it does not modify S if e is already in S [1]. The core of the algorithm is given by the iterative function $Order$ of Algorithm 1. At the beginning of the execution, it puts the unique node without predecessors and all its successors in the stack S_d. At each iteration, a node is removed from S_d and is processed by one of the three other functions depending on its type. This processing obeys to the following rules: if a node u is retrieved from S_d and u is marked, then, one will try to insert a new decomposable subgraph in the reduction order (here denoted ORD). Otherwise, one continues with some Depth First Search steps in order to explore the graph.

[1]For the implementation, we create for each stack a special marker that is set on each node when it is put in the stack

Algorithm 1 Compute the reduction order ORD

```
function ORDER(G_o, r, ORD, S_d, S_t, S_e)
    Mark r; Push r and all its successors in S_d
    while S_d is not empty do
        Pop(u)
        if u is an operation then
            ORDER_OPERATION(G_o,u, ORD, S_d, S_t, S_e)
        else if u is an opening connector then
            ORDER_OPEN(G_o,u, ORD, S_d, S_t, S_e)
        else if u is an closing connector then '
            ORDER_CLOS(G_o,u, ORD, S_d, S_t, S_e)
        end if
    end while
end function
function ORDER_OPERATION(G_o, u, ORD, S_d, S_t, S_e)
    if u is marked then
        if next(u) is not a closing connector or is not null then
            Insert(ORD, (u, next(u)))
        end if
    else if next(u) is a closing connector then
        Push_x(S_e, next(u)); Push_x(S_t, next(next(u)))
    else if next(u) is not null then
        Mark u; Push(S_d, u); Push(S_d, next(u))
    end if
end function
function ORDER_OPEN(G_o, u, ORD, S_d, S_t, S_e)
    if u is marked then
        Pop(S_e, w); Insert(ORD, (u, w))
        if next(w) is not a closing connector or null then
            Insert(ORD, (u, next(w)))
        end if
        Pop(S_t, v); Push(S_d, v)
    else
        Mark u; Push(S_d, u)
        Push all successors of u in S_d
    end if
end function
function ORDER_CLOS(G_o, u, ORD, S_d, S_t, S_e)
    Push_x(S_e, u); Push_x(S_t, next(u))
end function
```

Let us suppose that a marked connector is retrieved from S_d at a moment in the execution. The function ORDER is designed such as to guarantee at this moment that if we apply completely the built partial reduction order ORD, u will the root of an elementary subgraph whose leaf is in a node w the top of S_e. We will then insert (u, w) in the reduction order. Since after the reduction of (u, w), we will have to reduce the resulting node with the successor of $next(w)$, we will also insert $(u, next(w))$ if $next(w)$ is not a closing connector or null.

A marked operation u can also be retrieved at a moment from S_d. At this stage, the design of the ORDER function guarantees that if we apply the partial reduction order ORD, the successor of u will be a task. In this case, we will insert $(u, next(u))$ in the reduction order. Other important aspects of the algorithm are the management of closing connectors and the insertion of a graph in the reduction order. We discuss them below.

When we are exploring successors of an opening connec-

tor u in the stack S_d, if at a moment a closing connector is discovered, we will put it in the stack S_e. The algorithm is designed such as to guarantee that when we will retrieve u from S_d, on the top of S_e, we will have the corresponding closing connector. The intermediary stack S_t is also used for managing closing connectors. The idea is that when a closing connector v is encountered, a backtrack for identifying the corresponding opening connector u (in order to set an elementary subgraph (u, v) in ORD) will be done. A problem with this backtracking is that the successors of the encountered closing connector must still be explored. Thus, before backtracking, this successor is saved in the stack S_t. After treating u, the exploration continues with it.

Finally for the insertion of subgraphs in the reduction order ORD, the $Insert$ function is used. Given the couple (r, l) denoting the root and the leaf of a subgraph, this function will insert it in ORD after all couples (u, r) in ORD but before any couple (v, l). This choice ensures that when we will have to reduce a subgraph (r, l), where r is a task, l will be a reduced node. It is important to notice that the insert function can proceed in always inserting each couple (u, r) at the end of ORD. We need eventually to revert order only when an insertion must be done with an opening connector.

On Fig. 4, we give an example of decomposable graph. If we apply the Algorithm 1 on this graph, then the order generated at the end is: $\langle (H, K),\ (E, G),\ (c31, c32),\ (A, c31),\ (c41, c42),\ (B, c41),\ (c21, c22),\ (c21, J),\ (c11, c12),\ (c11, I)\rangle$. The importance of the $Insert$ function can be seen here. Indeed, in interverting the position of the couples $(A, c31)$ and $(c31, c32)$, we will have difficulties when applying reduction.

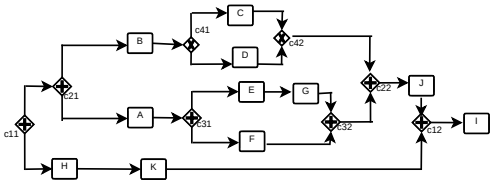


Figure 4: Example of decomposable graph

We showed how to build a reduction order. In the next section, we propose an analysis of our two phases approach for reduction.

E. Analysis

The first stage of our reduction approach consists in executing Algorithm 1. Its memory complexity is dominated by the storage of the reduction order $\langle G_1, \ldots, G_n\rangle$. Since $n \leq |V \cup C|$, we conclude that we use, at most, $O(|V \cup C|)$ space in the worst case. The main function in Algorithm 1 is the ORDER function. Its time complexity is dominated by two actions : 1) the number of times an element is removed from the stack S_d. 2) The number of time an element is put in it. Each node is put in S_d two times at most. In the first time it is not marked and in the second it is. This implies that this node can also be removed from the stack only twice: when it is not marked and when it is. We conclude that the time complexity of ORDER is in $O(2|V \cup C|)$. Given a reduction order, the second stage of our approach algorithm consists in using it to compute the SRT. We proceed for this iteratively. At the iteration i, the subgraph G_i is chosen and its SRT is computed using the rules defined in Table I. A new node is created with this associated SRT and G_i is replaced by this latter one. Its is easy to notice that at each iteration, we replace at most $|V \cup C|$ elements of G_o by a new node. Therefore, the reduction stage can be done in $O(n.|V \cup C|)$ or $O(|V \cup C|^2)$ since we have, at most, $n \leq |V \cup C|$ elementary subgraphs. Since the complexity of reduction when an order is available is in $O(|V \cup C|^2)$, we conclude that our two phases approach leads to a memory complexity in $O(|V \cup C|)$ while the time complexity is in $O(|V \cup C|^2)$.

In the result above, we assumed that given the operation graph $G_o = (V \cup C, E_{vc})$, E_{vc} has $O(|V \cup C|^2)$ elements. However, with our BPMN patterns, all operations and connectors will not be pairwise interconnected in a business process. It is reasonable to assume that we have $O(|V \cup C|)$ elements in E_{vc}. In this case, our algorithm has a time and memory complexity in $O(|V \cup C|)$. Let us notice that the SWR algorithm has, in this case, a time complexity in $O(|V \cup C|^2)$ [2].

At some points, our proposal for graph reduction in this work can be viewed as an iterative version of the recursive solution proposed in [6]. However, let us notice that our solution is based on the introduced notion of reduction order. In particular, we used it to explain why a correct result might be expected. In the work of [6], unstructured patterns are considered. We believe that in such cases, it is hard to obtain a definition of a reduction order that can always guarantee a reduction.

In the next section, we will consider the prediction of SRT with serialization.

V. Estimation of Service Response Time Estimation with serialization

Until now, we considered the prediction of QoS without serialization constraint. Let us now consider the examples given on Figure 5. In Fig. 5a), we have an elementary graph whose operations belong to different Web Services. There is no serialization required while executing this composition. In this case, our two phases approach can be used for SRT prediction. In the case in Fig. 5b) we have a business process with 4 operations A, B, C, D. Three of them (A, B, D) belong to the same Web Service $W1$. If we do not have three instances of $W1$, then some operations must wait for the termination of others. It is hard to take into account such situations in the graph reduction algorithm

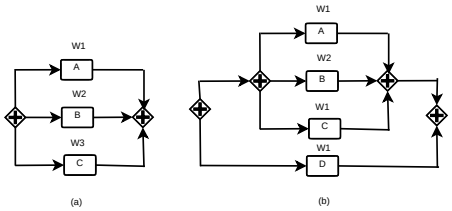


Figure 5: Example of operations unfolding. Each operations A, B, C is deployed on a Web Service $W1, W2, W3$

proposed before. For instance, when reducing the elementary subgraph composed of A, B, C, what must be the SRT for the operation A? Indeed, if we just have two W_1 instances, we have indeed multiple choices. We can set for instance $t_a + t_d$ by considering that the operation D will be executed before A, $t_a + t_c$ or t_a.

In this part, we deal with the prediction of SRT by taking into account this new constraint. Due to space limitations, we will restrict our study to decomposable graphs that contain only two types of patterns: the fork and sequence patterns. Let us notice that even with these restrictions, the serialization constraint remains. In the next we formalize the problem of QoS prediction with this additional constraint.

A. Modeling the problem of SRT prediction under serialization

In presence of conflictual execution of operations, an adequate selection of Web Service instance must be operated in the execution. In this part, we propose to estimate the minimal SRT that can be envisioned in presence of conflictual execution. The estimation of this time can be stated formally as follows:

*BPMN Fork-Sequence Scheduling Problem (*BFSP*)*

Instance: We have an operation graph $G_o = (V \cup C, E_{vc})$, for each operation $v_i \in V$ a mean execution time $t(v_i)$, a set of Web Service instances $W' = \{w'_1, \ldots, w'_u\}$, a function $f : V \longrightarrow W'$ giving for each operation the possible Web Service on which it can be executed.

Question: Find for each operation $v_i \in V$ a starting date $\tau(v_i) \in \mathbb{R}^+$ and a Web Service instance $\gamma(v_i) \in f(v_i)$ such that :

1) $Z = \max\{\tau(v_i) + t(v_i)\}$ is minimized ;
2) If $\gamma(v_i) = \gamma(v_j)$ then $\tau(v_i)+t(v_i) \leq \tau(v_j)$ or $\tau(v_j)+t(v_j) \leq \tau(v_i)$;
3) if v_i precedes v_j according to E_{vc}, then $\tau(v_i)+t(v_i) \leq \tau(v_j)$.

BFSP problem states the question of the best sequencing of the operations in order to minimize the maximal time (the Z above) at which an operation is finalized in a WSC. The constraint $2)$ states that we cannot execute two operations assigned on the same machine at the same moment. The constraint $3)$ states that we cannot execute an operation without the result of its predecessors. *BFSP* has some similitudes with the classical Web Service selection problem [6], [9] where we are looking for adequate Web Services for executing operations of a WSC. However, while in service selection, the goal is just to have an assignment of Web Services, in our case, we must, being given an assignment, define an execution order between operations. *BFSP* is on many aspects close to the multi-mode scheduling problem defined in [11]. The following result characterizes its hardness.

***Theorem** 2: BFSP* is NP-complete

Proof:

First, let us notice that it is simple to assert that given the dates τ_i for each operation and an assignment γ, one can estimate the maximal time for executing operations and check in polynomial time if it is a valid solution for *BFSP*. Thus, the problem is in the NP class. For showing the NP-completeness, We will proceed by reduction to the multiprocessor scheduling problem [11]. The idea of the reduction is that the multiprocessor scheduling problem correspond to *BFSP* when G_o is an elementary fork graph. The multiprocessor scheduling problem can be defined as follows: Given a set of n independent tasks $T_1, \ldots, T_n$, m identical resources, a time duration $l(T_i)$ that states for each task T_i its execution time on 1 resource, the objective is to find a n dates $d_1, \ldots, d_n$ for executing the tasks such that more than m tasks are not executed concurrently at a given time instant and $\max_{1 \leq i \leq n} \{d_i + l(T_i)\}$ is minimized.

We can solve any instance of this problem by associating to it a *BFSP* instance. This instance is defined as follows. We set the graph $G_o = (V \cup C, E_{vc})$ of the *BFSP* instance as an elementary fork graph whose internal nodes are operations v_i. Each v_i here corresponds to a task T_i. We define $W' = \{w'_1, \ldots, w'_m\}$ such that each w'_i corresponds to a resource. We define $f : V \longrightarrow W'$ as $f(v_i) = W'$ (that is each operation can be executed on any instance). For each operation, we set $t(v_i) = l(T_i)$. For solving a multiprocessor scheduling instance, we solve the associated *BFSP* instance and set for each task T_i of the multiprocessor instance, the value of d_i as the values $\tau(v_i)$ obtained from the *BFSP* instance.

It is easy to notice that if we have a solution for the associated *BFSP* instance, we can easily notice that in it, there is not more that m operations executed at a given date. Thus, we have a solution for the multiprocessor scheduling instance. Reciprocally, if we have a solution for the multiprocessor scheduling instance, then we have a solution for the associated *BFSP* instance in setting $\tau(v_i) = d_i$.

■

Since *BFSP* is NP-complete, it is hard to find a solution in polynomial time for it unless $P = NP$. In the sequel, we will propose an integer programming model for solving it.

B. Integer programming model for SRT prediction under serialization

The proposed model is based on three types of variables. These are the variables τ_i that contain real values stating the starting time of each operation $v_i \in V$. We have the variables γ_{ik} stating whether or not the operation v_i will be executed with the web service instance w'_k and the variables x_{ija} that we use to avoid cases where two operations v_i and v_j (with $i < j$) assigned to a same web service instance are executed during the same time period. For each operation v_i, we denote by $Pred(v_i)$ the set of operations that precede it. We propose the following integer programming model:

Minimize τ_{max}

Subject to:

1) $\tau_i \geq 0$; $\forall v_i$
2) $\gamma_{ik} \in \{0,1\}$; $\forall v_i, w'_k \in f(v_i)$
3) $\sum_{w'_k \in f(v_i)} \gamma_{ik} = 1$; $\forall v_i$
4) $\tau_j + t(v_j) \leq \tau_i$; $\forall v_j \in Pred(v_i)$
5) $x_{ija} \in \{0,1\}$; $\forall v_i, v_j, a \in \{1,2\}, i < j$
6) $x_{ij1} + x_{ij2} \leq 1$; $\forall v_i, v_j, i < j$
7) $x_{ij1} + x_{ij2} \geq \gamma_{ik} + \gamma_{jk} - 1$; $\forall v_i, v_j, i < j$, $w'_k \in f(v_i) \cap f(v_j)$
8) $\tau_i + t(v_i) - M(1 - x_{ij1}) \leq \tau_j$; $\forall v_i, v_j, i < j$
9) $\tau_j + t(v_j) - M(1 - x_{ij2}) \leq \tau_i$; $\forall v_i, v_j, i < j$
10) $M = \sum_{v_i \in V} t(v_i)$
11) $\tau_{max} \geq \tau_i + t(v_i)$; $\forall v_i$

In this formulation, the constraints 2 and 3 ensure that each operation will be assigned to one web service instance on which it can be executed. The constraint 4 ensures that an operation cannot be executed before its predecessors. The constraints $5, 6, 7$ define the possible values of x_{ija}. They guarantee that one of the values x_{ij1} or x_{ij2} will be equal 1 if v_i and v_j are assigned to the same web service instance. If $x_{ij1} = 1$ then from constraint 8 we know that the operation v_j cannot start before v_i. If $x_{ij1} = 0$, then if the web services instance assignment of v_i and v_j are conflicting, we necessary have $x_{ij2} = 1$. In this case, the operation v_j is executed before v_i.

One can notice that this model comprises a polynomial number of variables and constraints. Therefore, the SRT prediction under serialization can be made by solving the corresponding integer formulation using a classical integer programming solver and then returning the value of τ_{max}.

VI. Conclusion

In this paper, we investigated the question of Service Response Time prediction of a Web Service Composition using the graph reduction technique. We propose a fast algorithm for graph reduction and analyzed its complexity. We also analyze the prediction of Service Response Time when taking into account the unfolding of a Web Service Composition. In these cases, we showed that the SRT prediction can lead to an NP-complete problem. We propose in such cases an integer programming model for predicting the SRT. For continuing this work, we envision to consider the prediction of Service Response Time with a largest set of BPMN patterns. We also envision to consider additional delay time in the execution of a composition due for example to synchronization or information routing between operations.

Acknowledgment

This research was funded by HP Brasil under the Baile Project and from the European Communitys Seventh Framework Programme FP7/2007-2013 under grant agreement number 257178 (project CHOReOS - Large Scale Choreographies for the Future Internet). Yanik Ngoko is supported by the FAPESP foundation of the State of São Paulo

References

[1] M. Weske, *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.

[2] J. Cardoso, A. Sheth, J. Miller, J. Arnold, and K. Kochut, "Quality of service for workflows and web service processes," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 3, pp. 281 – 308, 2004.

[3] M. Jaeger, G. Rojec-Goldmann, and G. Muhl, "Qos aggregation for web service composition using workflow patterns," in *Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International*, sept. 2004, pp. 149 – 159.

[4] P. Puschner and A. Schedl, "Computing maximum task execution times - a graph-based approach," *Journal of Real-Time Systems*, vol. 13, pp. 67–91, 1997.

[5] S.-Y. Hwang, H. Wang, J. Tang, and J. Srivastava, "A probabilistic approach to modeling and estimating the qos of web-services-based workflows," *Information Sciences*, vol. 177, no. 23, pp. 5484 – 5503, 2007.

[6] H. Zheng, J. Yang, and W. Zhao, "Qos analysis and service selection for composite services," in *Services Computing (SCC), 2010 IEEE International Conference on*, july 2010, pp. 122 –129.

[7] K. R. Braghetto, J. E. Ferreira, and J.-M. Vincent, "Performance evaluation of resource-aware business processes using stochastic automata network," *International Journal of Innovative Computing, Information and Control*, p. to appear.

[8] Y. Xia, H. P. Wang, Y. Huang, and L. Yuan, "A stochastic model for workflow qos evaluation," *Sci. Program.*, vol. 14, pp. 251–265, December 2006.

[9] L. Zeng, B. Benatallah, A. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *Software Engineering, IEEE Transactions on*, vol. 30, no. 5, pp. 311 – 327, may 2004.

[10] A. Polyvyanyy, L. García-Bañuelos, D. Fahland, and M. Weske, “Maximal structuring of acyclic process models,” *CoRR*, vol. abs/1108.2384, 2011.

[11] L. Bianco, P. Dell’Olmo, and M. Speranza, “Scheduling independent tasks with multiple modes,” *Discrete Applied Mathematics*, vol. 62, no. 13, pp. 35 – 50, 1995.