Dynamic Selection for Service Composition Based on Temporal and QoS Constraints

Ikbel Guidara, Imane Al Jaouhari, Nawal Guermouche

Dynamic Selection for Service Composition based on Temporal and QoS Constraints

Ikbel Guidara[1,2], Imane Al Jaouhari[1] and Nawal Guermouche[1]
[1] LAAS-CNRS, Université de Toulouse, CNRS, UT1, INSA, Toulouse, France
[2]ReDCAD Laboratory, University of Sfax, ENIS, B.P. 1173, 3038 Sfax, Tunisia
iguidara@laas.fr, imane.al.jaouhari@gmail.com, nguermou@laas.fr

***Abstract*—To implement abstract business processes, elementary services are selected for each abstract task. Because of uncertainties of Quality of Service (QoS) values during execution, services may become faulty and cause the violation of end-to-end awaited constraints. Additionally, due to the dynamic nature of service systems, several environment changes may occur at run-time. In fact, services can join or leave the system or change their offerings. To deal with possible changes and maintain the feasibility of the selected solution, enabling dynamic service selection during execution is essential. This is not a trivial task especially in the presence of several constraints and dependencies between services namely QoS and temporal constraints. Existing approaches do not consider the specificities of temporal properties and usually handle violations after they have occurred. In this paper, a novel proactive dynamic service selection approach is proposed to deal with changes during execution while considering both QoS and temporal constraints. Experiments show that, by using our approach, faults can be successfully handled in a reasonable time while guaranteeing overall constraints.**

***Index Terms*—Dynamic Service Selection; Dynamic Service Systems; Uncertainties; Quality of Service; Temporal Properties**

I. Introduction

QoS (Quality of Service) based service selection is one of the important features of the SOC (Service Oriented Computing) paradigm. It allows to select adequate services to build compositions so that abstract business processes can be implemented. Service selection has gained main attention in the literature. Several attempts proposed strategies to select services at request time. These approaches select either the optimal or a near-to-optimal solution to obtain an ex-ante service composition [1], [2], [3]. During execution, the selected solution may deviate from the estimated values. In fact, since service-oriented systems are very likely to be executed in uncertain and dynamic environments, several changes can affect the selected services and may lead to the violation of end-to-end global constraints. In addition to changes that can be observed on the selected services, due to the dynamic nature of the execution environments, several changes can occur on the system when executing services. These changes may also have several impacts on the selected solution.

Therefore, enabling dynamic service selection to support the development of reliable service based processes is crucial. The aim is to adjust the selected service combination during the execution in response to the different changes to prevent violations that might occur. In real-world applications, QoS values of services may depend on the time (i.e., time-dependent QoS) [4], [5]. For instance the response time of a service can be longer during business hours. In addition, several temporal dependencies can exist in order to get control over the execution of processes. These constraints and dependencies make the dynamic service selection problem more complex.

Several selection approaches have been proposed to adapt service compositions during execution. A popular way consists on re-selecting all non-executed services [6], [1], [7]. This approach can be time consuming and usually requires the interruption of the execution which is highly undesirable. Moreover, these approaches suffer from poor scalability due to the consideration of all candidate services for each task. Other proposals identify backup solutions during the initial selection [8], [9], [10]. This is not suitable in highly dynamic environments. Most of current approaches take adaptation actions only for corrective purposes and do not provide techniques to enhance the selected composition at run-time (e.g., in response to the availability of a new better service). Moreover, they usually delay the reaction to changes until the failed service is executed which may lead to the inability to find a feasible solution. Furthermore, time-dependent QoS and temporal properties are not considered.

To overcome the limitations of existing approaches, in this paper, we propose a dynamic time-aware service selection approach. The proposed approach is proactive since it reacts to changes as soon as they occur in order to prevent possible violations at run-time. Reacting to changes at earlier stages allows to minimize the interruption time of the execution and to increase the likelihood of finding a feasible recovery. Our approach caters for both service violations and environment changes that can be observed at different stages of the execution. In contrast to existing approaches, in this paper, in addition to QoS offerings, we consider temporal constraints and their dependencies. Experiments show that the proposed approach allows to handle violations and changes successfully in a reasonable time without deteriorating overall constraints.

The rest of the paper is organized as follows. In the next section, we describe a motivating example. Section III discusses related work. In Section IV, we present the selection model. We detail our dynamic service selection approach in Section V. In Section VI, we evaluate the performance of the proposed approach. Finally, Section VII gives conclusions and future work.

II. Motivating Scenario

In this section, we provide a motivating scenario of device manufacturing. The corresponding business process is shown in Figure 1.

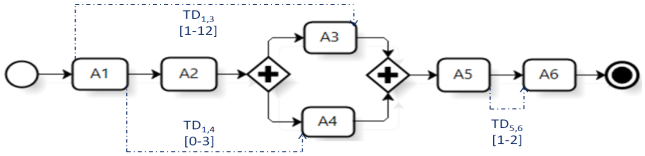

Fig. 1. The business process for the production of an electronic device

The production process starts by receiving and writing technical reports (A1). Then, the adequate model of the device is designed (A2). After that, the manufacturing of the electronic and peripheral parts is executed in parallel (A3 and A4). After manufacturing the various parts, these latter are assembled to deliver the final product (A5). Finally, the product is tested before delivering (A6). To increase its market share and profitability and gain control over the production time, the enterprise sets some constraints. In our example, we can identify the following constraints: (1) $TD_{1,3}$: The manufacturing of electronic parts has to finish no earlier than one time unit and no later than 12 time units after the receipt of the order. (2) $TD_{1,4}$: The manufacturing of peripheral parts starts no later than 3 time units after the writing of the technical reports. (3) $TD_{5,6}$: The test of the final product has to start no earlier than one time unit and no later than 2 time units before the assembly of all parts.

We suppose that three global constraints are associated with the process: (1) the cost must not exceed 68 cost units, (2) the duration of the execution must be less than 13 time units, and (3) the finish time of the process must not exceed 10 p.m (i.e., 22 units of time in our example). To be implemented, each abstract activity has a set of functionally similar concrete candidate services (Figure 2). Some services offer different QoS values according to their temporal properties. For instance, when the service S_{13} is available from 9 to 13, it offers a duration of 3 time units and a cost of 23 units (i.e., the service instance S_{131}). When it is available from 14 to 20, it offers an execution duration equals to 4 with a cost of 19 units (i.e., the service instance S_{132}).

	Avail	Dur	Cost		Avail	Dur	Cost		Avail	Dur	Cost
S_{111}	[8, 15]	4	12	S_{211}	[12, 17]	1	10	S_{311}	[13, 22]	4	12
S_{121}	[17, 21]	2	8	S_{221}	[7, 13]	4	12	S_{321}	[8, 14]	6	12
S_{131}	[9, 13]	3	23	S_{231}	[9, 12]	2	17	S_{331}	[10, 17]	4	13
S_{132}	[14, 20]	4	19	S_{232}	[14, 21]	1	15	S_{332}	[17, 23]	5	23
	Avail	**Dur**	**Cost**		**Avail**	**Dur**	**Cost**		**Avail**	**Dur**	**Cost**
S_{411}	[6, 13]	7	22	S_{511}	[7, 12]	1	8	S_{611}	[19, 22]	1	7
S_{412}	[13, 23]	9	18	S_{512}	[15, 20]	1	14				
S_{421}	[17, 24]	6	15	S_{521}	[8, 16]	3	20	S_{621}	[9, 13]	6	9
S_{431}	[12, 20]	4	12	S_{531}	[14, 22]	2	12	S_{622}	[14, 18]	4	10
								S_{631}	[8, 20]	1	6

Fig. 2. Candidate services of each abstract task

To better illustrate the usefulness of the proactive service re-selection, let's consider the following scenarios while assuming that the initial selected composition of services is CS^* = (S_{111}, S_{211}, S_{311}, S_{431}, S_{531}, S_{611}).

- *Scenario 1*: Suppose that during its execution, the service S_{211} takes 2 time units rather than 1. In this case, the global execution duration is not exceeded and the deadline is respected. Suppose now that another violation occurs when the service S_{611} is executing. This service takes 2 time units rather than 1. In this case, the service can not offer the expected cost and the overall deadline will be violated. Here, replacing the service S_{531} by S_{512} after the violation of the service S_{211} will result in a satisfactory solution even if the violation occurs on the service S_{611}. Thus, reacting to changes as soon as they occur allows minimizing the interruption time while increasing the chance to find a solution after violation.

- *Scenario 2*: Suppose that while executing the service S_{211}, the service S_{512} performs better with a cost equals to 10 rather than 14. In this case, replacing the service S_{531} by the service S_{512} will lead to a better optimality for the selected service composition. This helps in preventing proactively the violation of global constraints in case of future service's deviations.

III. Related Work

Depending on the time of the detection of failures and their recovery, selection approaches at run-time can be classified into *reactive* and *proactive* approaches. Reactive approaches [11], [6], [1] react to changes and deal with erroneous behaviors after their occurrence. This might lead to undesirable effects such as the inability to find a feasible solution. Moreover, the late reaction to changes might cause a significant interruption time during the execution of services which is highly undesirable mainly in time-sensitive applications. To deal with the limitations of reactive strategies, some approaches adopt a proactive strategy. The goal is to anticipate required selection actions prior to the occurrence of possible violations [10], [12], [13]. The main advantage of this strategy is to increase the likelihood of finding a possible solution while enhancing the overall quality and avoiding the interruption of service execution due to for instance the invocation of a faulty service.

Different techniques have been proposed to adapt service compositions at run-time. An easy way is by triggering re-planning actions for one or more non-executed tasks such that all global constraints are fulfilled. Some approaches adopt a *global service re-selection* to select services for all non-executed tasks [6], [1], [7]. Usually, the same selection algorithm, used to select the initial combination of services, is applied while considering the values of the executed services. Another global re-selection approach is proposed in [14] to deal with uncertainties of response time during execution. A Mixed Integer Programming (MIP) model is defined to adjust the combination of services when a deviation or a violation is observed. In this work, an optimized extended service binding is proposed to restrict the number of services considered in the reconfiguration phase. One issue of this work is that it considers only one quality attribute that should be optimized.

Other works apply a *partial service re-selection* to re-select only a subset of services. In [15], Lin et al. define reconfiguration regions to adapt the selected combination of services. In this work, a reconfiguration region is identified for each faulty service. These regions are enlarged using a distance measure until a satisfactory solution is found. However, to re-select services, all potential candidate services are considered which may lead to a costly adaptation of services. Additionally, reconfiguration actions are taken only when a violation occurs without preventing possible violations. A similar work that adopts a partial re-selection strategy is presented in [16] while considering only a small set of alternative services for each task. Authors preselect two services for each business task based on a distance measure with respect to the primary selected services. Considering only two services may lead to the inability to find a feasible solution in some cases based on the already executed services. Moreover, the measures used to select alternative services are applied locally for each abstract task and do not consider dependencies between alternative services of the different tasks. Service re-selection approaches generally require the interruption of the execution and can not handle environment changes.

Some researches resort to the use of *backup solutions*. Backup solutions are identified prior to the execution of the service composition so that, if a problem occurs, these solutions can be used to maintain the successful execution of services. In [8], Yu et al. propose an offline algorithm to adapt to service changes. The proposed algorithm identifies a secondary path from each service to the end service so that if a service becomes faulty at run-time, the secondary path from this service will be used to repair the combination of services. However, only one failure can be handled which is not appropriate in highly uncertain and dynamic environments. Dai et al. [10] propose to switch the execution to a backup solution. Despite the early reaction to changes, only services that succeed the failed service are considered. However, in some cases including also non-executed services that precede the affected service might lead to a better solution. In these approaches, all backup solutions are determined prior to the execution without taking into account environment changes. Considering static backup solutions at request time is not a practical solution when dealing with highly uncertain and dynamic environment and heavily constrained problems. For instance, it may be the case when no predefined plan is feasible even though a solution to the problem does exist.

Most of current approaches do not consider specific characteristics related to the presence of temporal properties.

IV. Service Selection Specification

Each business process is composed by a set of abstract tasks $\mathcal{A} = \{A_1, ..., A_n\}$. These tasks depend on each other by a set of *structural dependencies*. In addition, business designers can also require supplementary *inter-task temporal dependencies*. These latter define constraints related to the start and the finish time of business tasks. They express dependencies between two directly or indirectly succeeding tasks (e.g., a task *A* that has to start no earlier than *v* time units after the start of a task *B*). We denote by $\mathcal{TD}$ the set of temporal dependencies. Each temporal dependency $td_{ik} \in \mathcal{TD}$ between two tasks A_i and A_k is denoted by $td_{ik}(TP, D_{ik}^{min}, D_{ik}^{max})$ with TP is the type of the dependency (e.g., Finish-to-Start (FS), Start-to-Finish (SF)) and D_{ik}^{min} and D_{ik}^{max} denote the minimum and the maximum duration values between A_i and A_k, respectively.

For simplicity, we denote by s_{ij} a candidate service instance for the task A_i. Each service has a quality value $Q(s_{ij}, q_y)$ for each quality attribute $q_y \in \mathcal{QS}$ (with $\mathcal{QS}$ denotes the set of quality attributes) and a time span during which it offers different QoS values (Figure 2). The start and the finish time of each time span are denoted by $t_{s_{ij}}^{min}$ and $t_{s_{ij}}^{max}$, respectively. For each selection problem, a set of *global constraints* is required by users. A global constraint for a QoS attribute q_y is denoted by $Q(q_y)$. For simplicity, in this paper, we consider only negative QoS attributes (whose values need to be minimized).

The successful execution of the business process requires that the values of selected services are compliant with all constraints. In this paper, we assume that a *primary service combination* is already identified using existing static approaches [2], [3]. In [2], we proposed an optimal service selection approach to select the best combination of services while considering time-dependent QoS and temporal properties. This work has been extended in [3] by presenting a heuristic time-aware service selection approach to efficiently select a close-to-optimal solution in order to deal with large selection problems. In the following, we denote by $CS^* = \{s_1^s, ..., s_i^s, ..., s_n^s\}$ the selected solution with s_i^s is the selected service for the task A_i. Each selected service s_i^s has a set of quality values as well as two temporal values which are specified considering the set of all selected services: the estimated start time and the estimated finish time denoted respectively by st_i and ft_i. Table I presents some extra notations used through this paper.

TABLE I
NOTATIONS USED THROUGHOUT THE PAPER.

Notation	Description
m	The number of quality attributes
q_y	The y^{th} quality attribute with $1 \leq y \leq m$
W_y	The weight of q_y given by the user
$Q(A_i, q_y)^{min}$ $Q(A_i, q_y)^{max}$	Minimum and maximum values of q_y for the task A_i, respectively
$Q(q_y)^{min}$ $Q(q_y)^{max}$	Minimum and maximum aggregated values of q_y, respectively
$Q(A_i, q_y)$	The value of q_y of the selected service for A_i
Agg	The aggregation function that derives the quality values of the composite service from the quality values of its component services
$Q(CS, q_y)$	The value of q_y of the composite service CS with $Q(CS, q) = Agg_{A_i \in \mathcal{A}}(Q(A_i, q))$
$\mathcal{P}d(A_i)$	The set of immediate predecessors of the task A_i

V. Dynamic Service Selection Approach

The selection of a primary service combination is based on estimated values [2], [3]. During execution, actual values may deviate from the estimated ones which may cause the violation of business and global constraints. To ensure the reliable execution of the different business tasks while guaranteeing the

satisfaction of end-to-end global constraints, service processes need to continuously react to varying environmental conditions during execution. In this section, we present our dynamic selection approach to enable service composition adaptation each time new pertinent events arise.

A. Identifying Backup Services

To enhance the efficiency of our approach, we identify a set of alternative services for each abstract task. Thus, a local selection can be easily applied to substitute the failed service by one alternative service. In contrast to existing approaches that pre-select statically alternative services during the initial service selection (e.g., [16], [14]), in our approach, alternative services are dynamically identified. In fact, alternative services identified during the initial selection may become no more pertinent when considering the real information during execution. Thus, in our approach, we argue that alternative services must be updated and re-identified during execution each time a change occurs in the selected solution.

1) Execution Thresholds: To identify the most pertinent services for each task, first, we compute *maximum thresholds* that once they are exceeded, global constraints are guaranteed to be violated. These thresholds are computed for each task based on both QoS and temporal constraints. The main idea is to compute the maximum allowed values for each task while considering the values of the selected services for the remaining tasks. For instance, if we consider the example presented in Section II, maximum thresholds for the cost attribute of the tasks A_1, A_2, A_3, A_4, A_5 and A_6 are equal to 15, 13, 15, 15, 15 and 10, respectively, considering the selected combination of services CS^* = (S_{111}, S_{211}, S_{311}, S_{431}, S_{531}, S_{611}). Consequently, if for instance, the service S_{111} of the first task changes its cost value to 16 rather than 12, the global execution duration will be violated. To compute maximum thresholds for the duration attribute as well as maximum temporal thresholds (i.e., the latest start and finish time) for each task, we propose the model from (1) to (10).

$$\text{maximize} \quad Q(A_b, dur) \tag{1}$$

$$Agg_{A_i \in \mathcal{A}}(Q(A_i, dur)) \leq Q(dur), \forall A_i \in \mathcal{A} \tag{2}$$

$$ft_n \leq deadline \tag{3}$$

$$ft_i \leq st_k, \forall A_k \in \mathcal{A}, A_i \in \mathcal{P}d(A_k) \tag{4}$$

$$Q(A_i, dur) = Q(s_i^s, dur), \forall A_i \in \mathcal{A}, i \neq b \tag{5}$$

$$st_i + Q(s_i^s, dur) = ft_i, \forall A_i \in \mathcal{A}, i \neq b \tag{6}$$

$$ft_i + D_{ik}^{min} \leq st_k, \ \forall \ td_{ik}(FS, D_{ik}^{min}, D_{ik}^{max}) \in \mathcal{TD} \tag{7}$$

$$st_k \leq ft_i + D_{ik}^{max}, \ \forall \ td_{ik}(FS, D_{ik}^{min}, D_{ik}^{max}) \in \mathcal{TD} \tag{8}$$

$$Q(A_b, dur) \in [Q(s_b^s, dur), Q(dur)] \tag{9}$$

$$st_i, ft_i \in [t_{s_i^s}^{min}, t_{s_i^s}^{max}], \forall A_i \in \mathcal{A} \tag{10}$$

This model is applied for each task $A_b \in \mathcal{A}$. Maximum duration threshold of each task A_b is equal to its maximum allowed duration value (i.e., $Q(A_b, dur)$). The objective function (1) allows maximizing the duration value of the task A_b. Constraint (2) is used to guarantee that the global duration constraint is satisfied. Here, we do not give details about the aggregation function Agg. Further details can be found in our previous work [17]. To guarantee the satisfaction of the deadline, we use the Constraint (3). Constraint (4) deals with dependencies between tasks. The duration of each task $A_i \in \mathcal{A}, i \neq b$ is equal to the duration of its selected service (Constraint (5)). Moreover, the finish time of each task $A_i \in \mathcal{A}, i \neq b$ is equal to the sum of its start time and its duration (Constraint (6)). To deal with temporal dependencies, we use Constraints (7) and (8). For simplicity, we consider only finish-to-start dependencies. The domain of the maximum duration threshold of the task A_b is presented in Constraint (9). Finally, the start and the finish time of each task should be in the time interval of its selected service (Constraint (10)).

The set of maximum thresholds for each task A_i is denoted by $T_i^M = \langle T_{i1}^M, ..., T_{iy}^M, ..., T_{im+2}^M \rangle$ where T_{iy}^M denotes the maximum threshold for the attribute q_y, $\forall 1 \leq y \leq m$. T_{im+1}^M and T_{im+2}^M represent the maximum start and finish time of the task A_i, respectively. The latest finish time of the task A_i is equal to its finish time after applying the constraint optimization model from (1) to (10) (i.e., ft_i). Whereas, its latest start time is equal to $ft_i - Q(s_i^s, dur)$.

In addition, we identify a set of *intermediary thresholds* for each task in order to trigger selection actions before a violation of a global constraint occurs. The aim is to adjust the selected solution each time a deviation exceeds one intermediary threshold so that possible violations in the remaining non-executed tasks can be prevented. The set of intermediary thresholds for each task A_i is denoted by the vector $T_i^I = \langle T_{i1}^I, ..., T_{iy}^I, ..., T_{im+2}^I \rangle$. The intermediary threshold for each attribute can be computed by dividing the sum of the value of the selected service for this attribute and its corresponding maximum threshold by 2.

2) Alternative Services: The set of alternative services for each task $A_i \in \mathcal{A}$, denoted by $\mathcal{S}_{Supi}$, is the set of services that satisfy all maximum thresholds. Hence, a service s_{ij} is considered as pertinent to be part of the alternative set, if all its QoS values do not exceed maximum QoS thresholds and if it can start and finish its execution without exceeding the maximum temporal thresholds (i.e, $[t_{s_{ij}}^{min}, t_{s_{ij}}^{max}] \cap [st_i, ft_i] \geq Q(s_{ij}, dur)$) with st_i and ft_i are the time values computed using the model from (1) to (10). A service s_{ij} that satisfies all maximum thresholds of its corresponding task is denoted by (s_{ij} sat T_i^M). In what follows, $\neg(s_{ij}$ sat $T_i^M)$ denotes that at least one maximum threshold is not satisfied by the service s_{ij}. In our approach, alternative services are ranked according to their distance to the maximum thresholds. The service that has the largest distance (i.e., the largest score) to the maximum thresholds will be best ranked. The score of each service $s_{ij} \in S_{Supi}$, denoted by $score(s_{ij})$, is computed as follows with $T_{iy}^M \neq Q(A_i, q_y)^{min}$, $\forall 1 \leq y \leq m, \forall A_i \in \mathcal{A}$.

$$score(s_{ij}) = \sum_{y=1}^{m} W_y * \frac{T_{iy}^M - Q(s_{ij}, q_y)}{T_{iy}^M - Q(A_i, q_y)^{min}} \tag{11}$$

B. Handling Changes at Run-time

Changes can be divided into two categories: *changes in the services of the selected solution* (i.e., $s_i^s \in CS^*$) and *changes in the environment* (i.e., an *addition* of a new service, a *deletion* or a *modification in the values* of a candidate service).

1) Handling Changes in the Selected Solution: Algorithm 1 details our approach to handle changes in the selected services. If a deviation does not exceed the intermediary thresholds, it does not affect the selected solution (lines 4 and 5). If the deviation is between the intermediary and the maximum thresholds (line 6), our approach proceeds as follows: first, maximum thresholds and the set of alternative services are updated for each non-executed task considering the values of the already executed services (lines 7 and 8) where A_{ne} denote the set of non-executed tasks. We note that when updating temporal thresholds, the start and the finish time of the already executed services are considered in the model from (1) to (10). Moreover, the set of supplementary services is updated by identifying the new alternative services based on the new values of maximum thresholds and by computing the score of each service based on equation (11). Then, if there is at least one alternative service that has a score greater than that of the selected service, it will be considered in the selected solution (lines from 9 to 11) where s_i^1 denotes the first alternative service. The aim of this step is to avoid the accumulation of deviations during execution in order to prevent possible violations. If a deviation exceeds at least one maximum threshold or the selected service is no more available (line 12), then, the selected solution is no more satisfactory. In this case, all maximum thresholds and alternative services are updated for all non executed tasks (lines 13 and 14). If at least there is one service in the set of alternative services for a non-executed task, the selected service of this task will be substituted by the first alternative service (lines from 15 to 20). We note that the computation of thresholds and backup services can be applied in parallel for all tasks since they are independent from one task to another. In case no solution is found, the region-based service selection is applied (lines 21 and 22). This step is detailed in the next section. In all cases, if the selected solution is modified, all thresholds and alternative services will be updated (lines 23 and 24). It is worth noting that in our approach, if a violation that exceeds at least one intermediary threshold occurs in a selected but not yet executed service, we simply replace the selected service by the first alternative service (in case this latter has a better score). Steps in lines from 6 to 22 are applied only if the violation occurs in a service currently in execution.

2) Handling Changes in the Environment: Algorithm 2 details how we deal with environment changes during execution. If a service s_{ij} is added for a task A_i or a candidate service changes its values, then, if it satisfies all maximum thresholds, it will be added to $\mathcal{S}_{Supi}$ (lines from 3 to 5). If its score is better than that of the selected service, it will be selected in CS^* (lines from 6 to 8). If an alternative service is no more available, it will be removed from $\mathcal{S}_{Supi}$ (lines 9 and 10).

Algorithm 1 Handling Changes in the Selected Solution

1: **Input:** The service violation
2: **Output:** The new solution CS^* and alternative services
3: $Sol = \emptyset$
4: **if** (s_i^s sat T_i^I) **then**
5: $\quad Sol = CS^*$
6: **if** $\neg$(s_i^s sat T_i^I) and (s_i^s sat T_i^M) **then**
7: $\quad$ **for** each $A_i \in A_{ne}$ **do**
8: $\quad\quad$ *update*($T_i^M, \mathcal{S}_{Supi}$)
9: $\quad\quad$ **if** $score(s_i^1) > score(s_i^s)$ **then**
10: $\quad\quad\quad CS^* = CS^* \setminus \{s_i^s\} \cup \{s_i^1\}$
11: $\quad\quad\quad Sol = CS^*$
12: **if** $\neg$(s_i^s sat T_i^M) or s_i^s is no available **then**
13: $\quad$ **for** each $A_i \in A_{ne}$ **do**
14: $\quad\quad$ *update*($T_i^M, \mathcal{S}_{Supi}$)
15: $\quad$ **while** $Sol = \emptyset$ and $i \leq n$ **do**
16: $\quad\quad$ **if** $\mathcal{S}_{Supi} \neq \emptyset$ **then**
17: $\quad\quad\quad CS^* = CS^* \setminus \{s_i^s\} \cup \{s_i^1\}$
18: $\quad\quad\quad Sol = CS^*$
19: $\quad\quad$ **else**
20: $\quad\quad\quad i = i + 1$
21: $\quad$ **if** $Sol = \emptyset$ **then**
22: $\quad\quad Sol \leftarrow$ *regionBasedServiceSelection()*
23: **if** $Sol \neq \emptyset$ **then**
24: $\quad$ *update*($T_i^I, T_i^M, \mathcal{S}_{Supi}$)

Algorithm 2 Handling Changes in the Environment

1: **Input:** The environment change
2: **Output:** The new solution CS^* and alternative services
3: **if** Addition or modification of a service s_{ij} **then**
4: $\quad$ **if** (s_{ij} sat T_i^M) **then**
5: $\quad\quad$ *update*($\mathcal{S}_{Supi}$)
6: $\quad\quad$ **if** $score(s_i^1) > score(s_i^s)$ **then**
7: $\quad\quad\quad CS^* = CS^* \setminus \{s_i^s\} \cup \{s_i^1\}$
8: $\quad\quad\quad$ *update*($T_i^I, T_i^M, \mathcal{S}_{Supi}$)
9: **if** Deletion of a service s_{ij} **then**
10: $\quad \mathcal{S}_{Supi} = \mathcal{S}_{Supi} \setminus \{s_{ij}\}$

C. Region-based Service Re-selection

To avoid considering all non-executed services, we identify re-selection regions. The main idea is to include a small number of non-executed services in each region (hereafter denoted by $\mathcal{R}$) and then, expand the region until a solution is found. The identification of regions is based on [15]. The selection problem is formulated as a constraint optimization model (Constraints from (12) to (22)). The objective function (12) allows selecting the best solution. Only the set of alternative services before violation is considered in each region. To guarantee that only one service will be selected for each task, we add Constraint (14) with $a_{ij} = 1$ if the service s_{ij} is selected and 0 otherwise. In addition, to avoid searching for the corresponding global constraints for each region, we apply the selection algorithm for all tasks while assigning to each task that does not belong to $\mathcal{R}$, the originally selected service

before the violation (Constraints (15) and (16)). To guarantee the satisfaction of global constraints, we add Constraint (17). The start and finish time of each non-executed task belong to the time span of its selected service (Constraints from (18) to (20)). The start and the finish time of each executed task are equal to the time of its already executed service. Constraint (21) identifies the domain of the time interval of each task in $\mathcal{R}$. The start and the finish time of a non-executed task that does not belong to the selection region should belong to the time span of its already selected service (Constraint (22)). Finally, Constraint (23) guarantees that the overall deadline is satisfied. To handle structural and temporal dependencies between tasks, Constraints (4), (7) and (8) can be used.

$$\text{maximize} \quad \sum_{y=1}^{m} W_y * \frac{Q(q_y)^{max} - Q(CS, q_y)}{Q(q_y)^{max} - Q(q_y)^{min}} \tag{12}$$

$$Q(CS, q_y) = Agg_{A_i \in \mathcal{A}}(\sum_{s_{ij} \in \mathcal{S}_{Supi}} a_{ij} * Q(s_{ij}, q_y)), \forall q_y \in \mathcal{QS} \tag{13}$$

$$\sum_{s_{ij} \in \mathcal{S}_{Supi}} a_{ij} = 1, \forall A_i \in \mathcal{A}, a_{ij} \in \{0, 1\} \tag{14}$$

$$a_{ij} = 1, \forall A_i \notin \mathcal{R}, s_{ij} = s_i^s \tag{15}$$

$$a_{ij} = 0, \forall A_i \notin \mathcal{R}, s_{ij} \neq s_i^s \tag{16}$$

$$Q(CS, q_y) \leq Q(q_y), \forall q_y \in \mathcal{QS} \tag{17}$$

$$\sum_{s_{ij} \in \mathcal{S}_{Supi}} a_{ij} * t_{s_{ij}}^{min} \leq st_i, \forall A_i \in A_{ne} \tag{18}$$

$$st_i \leq \sum_{s_{ij} \in \mathcal{S}_{Supi}} a_{ij} * (t_{s_{ij}}^{max} - Q(s_{ij}, dur)), \forall A_i \in A_{ne} \tag{19}$$

$$ft_i = st_i + \sum_{s_{ij} \in \mathcal{S}_{Supi}} a_{ij} * Q(s_{ij}, dur), \forall A_i \in A_{ne} \tag{20}$$

$$st_i, ft_i \in [min_{s_{ij} \in \mathcal{S}_{Supi}}\{t_{s_{ij}}^{min}\}, max_{s_{ij} \in \mathcal{S}_{Supi}}\{t_{s_{ij}}^{max}\}], \forall A_i \in \mathcal{R} \tag{21}$$

$$st_i, ft_i \in [t_{s_i^s}^{min}, t_{s_i^s}^{max}], \forall A_i \notin \mathcal{R}, A_i \in A_{ne} \tag{22}$$

$$ft_n \leq deadline \tag{23}$$

VI. Performance Evaluation

In this section, we evaluate the performance of the proposed approach by studying its time complexity and analyzing experimental results based on simulation studies.

A. Complexity evaluation

The complexity of our approach is mainly affected by the complexity of the local selection (lines from 12 to 20 in Algorithm 1) and the region-based selection steps.

- *Local service selection:* The time complexity of the local selection stage depends on the complexity of the following steps: (i) computing thresholds, (ii) updating the set of alternative services and (iii) locally selecting new services. The computation of thresholds is independent on the number of services per tasks and the number of business tasks, since QoS and temporal thresholds for all tasks can be computed in parallel. Thus, the complexity of this step is $O(no)$ with *no* denotes the operation number of equation resolution of the proposed model. The updating of the set of alternative services consists on selecting and computing the scores of the services that satisfy the new thresholds which has a complexity of $O(s)$ (with s is the number of alternative services per task) and the ranking of the new set of alternative services which has a complexity equals to $O(ns)$ in the best case and $O(ns * log(ns))$ in the worst case with *ns* is the number of the new alternative services. For simplicity, we assume that the number of alternative services is the same for all tasks. The local selection has a complexity of $O(1)$ in the best case (i.e., the first non-executed task has a non empty set of alternative services) and a complexity of $O(e)$ in the worst case (i.e., only the last non-executed task has a non empty set of services) with e denotes the number of non-executed tasks (i.e., $e = \mid A_{ne} \mid$). Thus, the complexity of the local selection phase is equal to $O(no + s + ns * log(ns) + e) = O(ns * log(ns))$ in the worst case.

- *Region based service selection:* The complexity of this step is equal to $O(2^{r*s})$ with *r* is the number of tasks in the selection region $\mathcal{R}$. More specifically, it is equal to $O(2^{2*s})$ in the best case (i.e., in case when a solution to the selection problem is found when only two non-executed tasks are included in $\mathcal{R}$) and is equal to $O(2^{e*s})$ in the worst case (i.e., all non-executed tasks are included in $\mathcal{R}$).

In conclusion, in the worst case, the complexity of our approach is equal to $O(ns * log(ns))$ when a solution is found based on local selection and is equal to $O(2^{e*s})$ when a solution is found using the region based selection approach.

B. Experimental Results

To evaluate the performance of our approach, we conduct experiments using a complex business process composed of 10 abstract tasks. The structure of the process is generated randomly. The number of candidate services and QoS constraints is fixed to 500 and 5, respectively. Constraint optimization models are implemented using the constraint solver Choco[1].

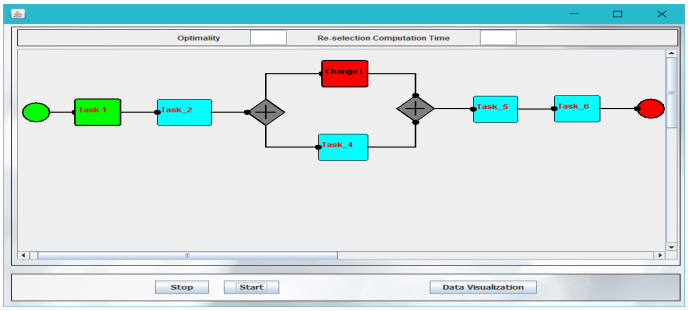

Fig. 3. Progress of the electronic device production business process

We have developed a simulation tool to perform the different test cases. This tool implements the proposed approach and manages faults and changes during execution. It also allows

[1]http://www.emn.fr/z-info/choco-solver/

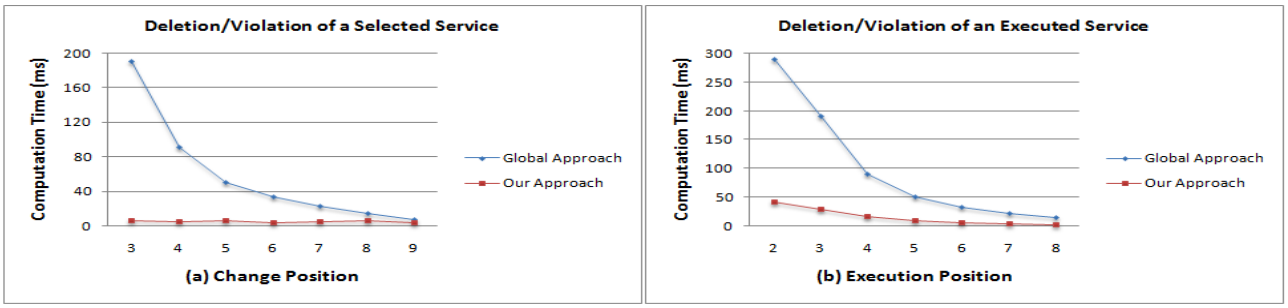

Fig. 4. Performance in terms of computation and interruption time

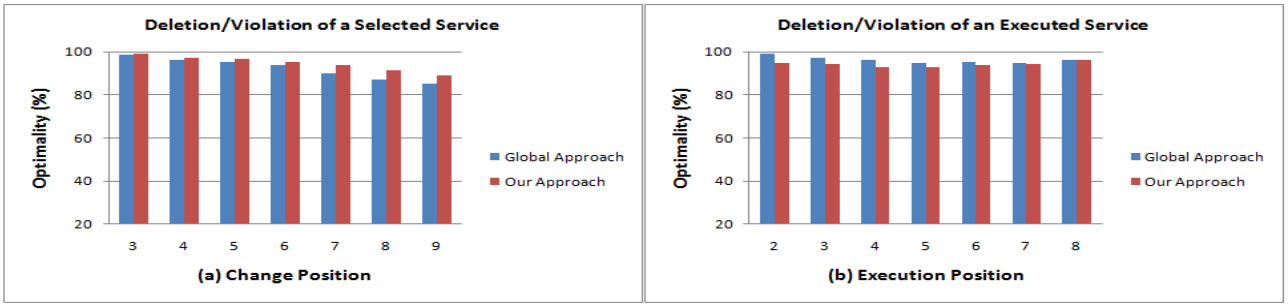

Fig. 5. Performance in terms of optimality

visualizing the execution progress of the different tasks as well as the different changes that might occur at run-time. Figure 3 shows a screenshot of the tool where the execution of the process presented in Section II is depicted. We can see that a change occurs in task A_3 (task in red) during the execution of the task A_1 (task in green). We study the performance of our approach in terms of the computation time, the optimality and the success rate comparing to the global approach in which a selection from scratch for all unexecuted tasks is applied [1].

First, we compare the computation time of the two approaches in the case of a deletion or a violation of a selected but not yet executed service (Figure 4(a)) and a deletion or a violation of a service currently in execution (Figure 4(b)). In these two cases, we suppose that the violation exceeds the maximum thresholds and thus a re-selection is mandatory to guarantee the satisfaction of global constraints. In Figure 4(a), we assume that the violation or the deletion of the selected service occurs after two tasks from the one being executed (e.g., while executing the first task, the violation occurs on the third task). Experiments reveal that the computation time of our approach is negligible compared to that of the global approach. This is mainly due to the fact that in our approach, a solution can be found based on local re-selection. However, the global approach delays the reaction to changes until the execution of the failed service. We note that unlike the global approach, our approach does not cause the interruption of the execution since the selection actions are taken as soon as the change occurs in parallel to the execution. In Figure 4(b), when the interruption of execution is required, our approach has a very small interruption time even when the change occurs at earlier stages. In fact, in some cases, our approach can find a solution based on local selection. Moreover, selecting services only for a specific region while considering a set of alternative services for each task allows enhancing the performance of our approach since it decreases the number of possible combinations that have to be compared. Experiments also show that the computation time of both approaches decreases when the position of the executed task get closer to the end of the process. This is due to the fact that the number of tasks that are considered in the selection process becomes smaller and the number of possible solutions decreases.

To evaluate the gain of utility of our approach, we compare its optimality with that of the global approach (Figures 5). The optimality of each approach is computed by dividing the utility value (equation (12)) of the obtained solution after changes by the utility value of the primary combination of services. As can be seen in Figure 5(a), our approach reaches better optimality in all cases since it allows early reaction to changes. Figure 5(b) indicates that in the case where the erroneous behavior is observed in the service currently in execution, our approach can produce a satisfactory optimality. Here, the global approach has better optimality since it considers all possible combinations of services in contrast to our approach that can find a solution by local selection.

Figure 6 (a) depicts the computation time in response to the number of changes which are added randomly at run-time. These changes include the addition, the modification and the deletion of alternative services as well as deviations in the selected services. All deviations are assumed to be less than the maximum thresholds. The positions of changes and deviations

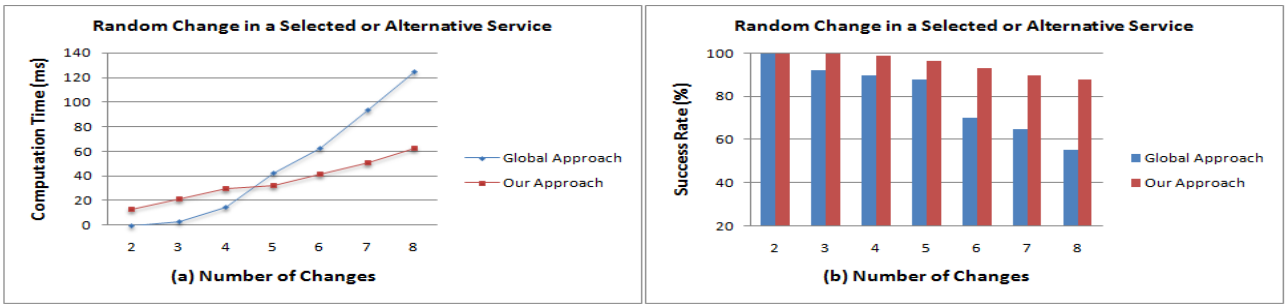

Fig. 6. Computation time and success rate versus number of changes

are generated randomly. Results show that although the computation time of the global approach is almost negligible when the number of changes is very small, it increases by increasing the number of changes. In contrast, the computation time of our approach is very small in most cases. This is explained by the fact that the global approach usually does not react to changes if the global constraints are not violated. If a global violation occurs, the execution is interrupted until a solution is found and usually a re-selection for all the non-executed portion of the business process is required which can be time consuming. However, our approach takes selection actions and enhance the selected solution even before the occurrence of a global violation. Figure 6 (b) indicates that our approach has better success rate since it reacts to changes as soon as they occur which increases the likelihood to find a solution. Whereas, when using the global approach, it might be the case where no solution is found after a global violation. This is due to the fact of delaying the reaction to changes after the execution of the failed service.

VII. Conclusion

In this paper, we proposed a dynamic service selection approach to support the development of reliable business processes. The goal is to tackle uncertainties and dynamic environments during execution. To do so, a set of thresholds is identified for each task to characterize the pertinence to trigger dynamic selection actions. To enhance the efficiency of our approach, we identify a set of alternative services for each task. Our approach differs from existing ones by the fact that thresholds and alternative services are updated during execution based on the values of the already executed services. Moreover, reactions to changes are made as soon as they occur in order to avoid execution interruption and increase the likelihood of finding a satisfactory solution. To reduce the complexity of the proposed approach, a local service selection is applied. Moreover, we propose a region based service selection approach to avoid re-selecting all non-executed services. Unlike existing work, our approach allows not only dealing with run-time deviations, but also it enables to enhance the selected service composition. Moreover, our approach considers time-dependent QoS associated with temporal constraints. To study the effectiveness of our approach, we implemented a simulation tool. The experiments show that our approach performs well in most cases by finding feasible solutions in a reasonable time with a satisfactory optimality. As a future work, we aim to extend the proposed approach to deal with several changes at the same time and propose strategies to handle potential conflicts between recovery actions. In addition, we intend to further compare our approach with other existing approaches based for example on backup solutions.

References

[1] D. Ardagna and B. Pernici, "Adaptive service composition in flexible processes," *IEEE Trans. Software Eng.*, vol. 33, pp. 369–384, 2007.

[2] I. Guidara, N. Guermouche, T. Chaari, S. Tazi, and M. Jmaiel, "Pruning based service selection approach under qos and temporal constraints." in *ICWS*, 2014, pp. 9–16.

[3] ——, "Heuristic based time-aware service selection approach," in *ICWS*, 2015, pp. 65–72.

[4] L. Chen, J. Yang, and L. Zhang, "Time based qos modeling and prediction for web services," in *ICSOC*, 2011, pp. 532–540.

[5] F. Wagner, A. Klein, B. Klöpper, F. Ishikawa, and S. Honiden, "Multi-objective service composition with time- and input-dependent qos," in *ICWS*, 2012, pp. 234–241.

[6] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani, "Qos-aware replanning of composite web services," in *ICWS*, 2005, pp. 121–129.

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

[8] T. Yu and K. Lin, "Adaptive algorithms for finding replacement services in autonomic distributed business processes," in *ISADS*, 2005, pp. 427–434.

[9] G. Chafle, P. Doshi, J. Harney, S. Mittal, and B. Srivastava, "Improved adaptation of web service compositions using value of changed information," in *ICWS*, 2007, pp. 784–791.

[10] Y. Dai, L. Yang, and B. Zhang, "Qos-driven self-healing web service composition based on performance prediction," *J. Comput. Sci. Technol.*, vol. 24, no. 2, pp. 250–261, 2009.

[11] S. Kalasapur, M. Kumar, and B. Shirazi, "Dynamic service composition in pervasive computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 18, no. 7, pp. 907–918, 2007.

[12] R. Aschoff and A. Zisman, "Qos-driven proactive adaptation of service composition," in *ICSOC*, 2011, pp. 421–435.

[13] A. Metzger, O. Sammodi, K. Pohl, and M. Rzepka, "Towards pro-active adaptation with confidence: augmenting service monitoring with online testing," in *ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, SEAMS*, 2010, pp. 20–28.

[14] R. Ramacher and L. Mönch, "Reliable service reconfiguration for time-critical service compositions," in *SCC*, 2013, pp. 184–191.

[15] K. Lin, J. Zhang, and Y. Zhai, "An efficient approach for service process reconfiguration in SOA with end-to-end qos constraints," in *CEC*, 2009, pp. 146–153.

[16] J. Li, D. Ma, X. Mei, H. Sun, and Z. Zheng, "Adaptive qos-aware service process reconfiguration," in *SCC*, 2011, pp. 282–289.

[17] I. Guidara, N. Guermouche, T. Chaari, M. Jmaiel, and S. Tazi, "Time-dependent qos aware best service combination selection," *Int. J. Web Service Res.*, vol. 12, no. 2, pp. 1–25, 2015.