Optimizing Time in Workflow-Based Business Processes When Handling Exceptions

Yuqun Zhang, Dewayne E. Perry

Center of Advanced Research in Software Engineering (ARiSE)
Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX, USA
Email: yuqun_zhang@utexas.edu, perry@mail.utexas.edu

Abstract. Dynamic exception handling is a major technique to ensure the quality of runtime business processes. In this paper, we study the methods to realize time optimization for workflow-based business processes when handling its runtime exceptions. By applying our data-centric modeling technique for workflow-based business processes, we explore the exception patterns that are possible to avoid jeopardizing the execution of business processes and analyze their respective impacts on time performance. We also develop algorithms to adjust the execution order of business activities dynamically for the exception patterns such that an optimize time performance can be realized.

1 Introduction

Dynamic exception handling techniques for runtime business processes have been well studied in the past decades: the authors of [1] [2] [3] propose solutions for the exceptions caused by missing deadlines of business activities. Predicability is provided in [4] [5] for their respective target business process properties. Time scale for each business activity is estimated in [6] for designing a schema to handle the dynamic assignment of business activity executions.

However, to our knowledge, dynamic exception handling techniques have rarely been proposed for business process optimization, which is usually realized by automatically improving business processes using quantitative measures of performance (objectives) in a predetermined manner [7].

In this paper, on the purpose of business process optimization, particularly time performance under runtime exceptions, data-centric dynamic exceptions handling approaches are developed based on our previous approaches of business process modeling [8] and static time optimization [9]. As in [10] [11], working time, defined as the total lapse time to complete a process unit (e.g., business activity, subprocess, cluster, etc.) is typically either known or estimated during the design phase of business processes. Nevertheless, potential conflict occurs when an execution of a business activity in runtime business processes demands real-time working time that deviates from the predetermined time (i.e., it encounters exceptions). This deviation might change the pattern of the time performance of

the overall business process, causing the original time optimality to be incorrect (in this paper for simplicity, time performance of the predefined business process is assumed to be optimized), and provide possibilities to dynamically optimize runtime time performance of the overall business process.

Our approach in this paper is introduced by illustrating the exception patterns that disable the predetermined optimized time performance of business processes yet possibly enable runtime business processes to continue to be executed and completed eventually. With respect to the data-centric perspectives, complete executions of business processes are reflected by successfully delivered data flows, that indicate exceptions under complete executions of business processes cannot hinder the data delivery within the business processes. Accordingly, three such exception patterns in these regards (partial data, halted data, and business activity insertion/deletion) are specified in our approach.

Partial data refers to the case where instead of all, only a partial amount of data is generated after the execution of business activities. Due to their real-time working time and the portion of the data deliveries, we provide intuitive methods to re-estimate the working time for the completion of the execution of these business activities as the basis of dynamic time optimization. Halted data indicates that data delivery is halted during the execution of business activities. Business activity insertion/deletion illustrates the circumstance that in runtime business processes, business activities that are not in the predetermined business processes need to be inserted or those which have not been executed in the predetermined business processes need to be deleted.

Algorithms are developed to approach optimized time performance of the overall business processes for each exception pattern. For the pattern of partial data and business activity deletion, it is as simple as to statically optimize time performance using our previous approach [9]. For the pattern of halted data, "best efforts" are provided to explore the possibility of executing the business activities that are supposed to be executed later in the subprocess (defined to be composed of business activities in execution order and bounded by their convergence and divergence relations) with the halted data. For the pattern of business activity insertion, business activities are inserted to the predetermined business processes in a way that their individual contributions to increasing the time performance of the overall business processes can be minimized. Moreover, an additional "backtracking" process is implemented to diminish their combinational contributions to increasing the time performance.

In sum, the major contributions of our approach in this paper are: *1)* it specifies the exception patterns in which the execution of business processes is possible completed, and studies their impacts on the time performance of business processes; *2)* it develops methods to approach optimized time performance for each of them.

The rest of the paper is organized as follows. Section II introduces related work. Section III revisits our data-centric business process modeling and static time optimization approaches. Section IV describes our approach of dynamically optimizing time performance of business processes. Section V discusses our conclusions and future work.

2 Related Work

Exception handling techniques are usually studied for runtime business processes. A number of the relevant approaches aim at handling the exceptions under the circumstances that deadlines of business activities are missed during runtime. The authors of [2] describe how information about time can be captured in workflow-based systems and provide various estimations about the working time of business activities, such that violations of deadlines assigned to activities and the overall business process can be avoided or reacted to when they occur. The authors of [3], based on history logs of a business process, build a time probability model for the indeterminacy of the execution time of tasks. In addition, they provide algorithms to detect the possible deadline violations by analyzing the models with the realistic workload of the associated resources. To predict a workflow's deadline, the authors of [5] decompose the selected structure of workflow, insert checkpoints into the decomposed activities, and propose the corresponding predicting algorithms.

Time is considered as a metric to evaluate the performance and quality of modeling techniques. Researchers of [12] [13] include time as the dynamic view to evaluate business process configuration. With a known working time of each business activity in [14], methods are developed to detect the degree of parallelism, that is defined as the number of the executing business activities at the same time. Researchers of [10] [11] implement empirical studies of software developments and conclude the factors that lead to the discrepancy of race time, defined as the time spent on actual work, and elapsed time, defined as additionally including the presence of interruption, blocking, and waiting periods. However, these researches fail to reason and solve how time performance can be improved.

3 Data-Centric Business Process Modeling and Static Time Optimization Approaches Revisited

In our previous research [8], a workflow-based business process is perceived to be a homogeneous combination of data flows, where a business activity is composed of a tuple of components (data, human actor, and atomic activity). A business activity is designed with one or more atomic activities being executed by identical human actor(s). The associated data is further modeled to indicate its states in data flows within a business activity: *initial_data, input_data, global_data, consumed_data, output_data,* and *final_data*, with their properties simply introduced as follows:

- *initial_data*: data that is injected to the data flows in its first appearance and triggers the execution of the associated business activity.
- *input_data*: data that is produced and delivered by the executions of other business activities and triggers the execution of the associated business activity.
- *global_data*: data that is injected independently from business processes and triggers the execution of the associated business activity.

- *consumed_data*: *input_data* or *initial_data* that is consumed after the execution of the associated business activity.
- *output_data*: data that is produced by the execution of the associated business activity.
- *final_data*: data that is produced by the execution of a business process and not consumed by any business activity, or a state to indicate the end of the execution of a business process; indicates the end of certain data flows.

We perceive *global_data* as a constant and do not consider its contribution to evaluating any property of a business process.

In our research of static time optimization approach [9], based on the design of business processes, business activities are relocated from their original subprocesses to other subprocesses to optimize time performance of the overall business process. The eligibility of this business activity relocation is defined as: given a business activity and a subprocess where the business activity is not originally located, the business activity is eligible for relocation in the business process when it can be **forwardly moveable, backwardly moveable**, or **loosely-parallelled moveable** to that subprocess.

The algorithm to optimize time performance of the overall business processes is subprocess-based, that is, for a subprocess that needs to optimize time performance, its eligible business activities are collected across the overall business process and sorted due to their contributions to optimizing its time performance. Eventually, they are selected to be relocated to the subprocess such that its time performance can be optimized. This process repeats to all the subprocesses that need to optimize their respective time performance and the optimized time performance of the overall business process is approached.

4 Approach Overview

In this section, we introduce our approach to optimize time performance for the overall business process when handling runtime exceptions. Our approach is categorized into two parts: introduction to exception patterns, where all the exception patterns that possibly allow the completion of the executions of the business processes are defined and their respective properties are analyzed; and algorithms of time optimization for each of the exception patterns.

4.1 Exception Patterns

Generally, for a business activity that is being executed in a longer real-time working time than the predefined working time, it would result in two possibilities: completion in finite time and incompletion in infinite time, that correspond to the exception patterns of partial data and halted data, respectively.

In addition, it is possible that during the execution of business processes, business activities need to be instantaneously inserted to or deleted from the predetermined business process. In our approach, these business activities are assumed to be designed with appropriate data types such that their insertions or deletions cannot jeopardize the data flows.

Partial Data In fact, the pattern of partial data includes the following scenarios: the execution of a business activity being completed with more time than expected and *full/partial output_data*, while they are perceived identical in terms of their impacts on time performance. Thereby in our approach, they are included as the partial data patterns.

Essentially, the pattern of partial data refers to the scenario that only a partial amount rather than the total expected amount of the *output_data* or the *final_data* is delivered after the execution of the associated business activities. In our approach, partial data implies the possibility of complete data by re-executing the associated business activities. Accordingly, assuming partial data occurs only for one data item, the working time of the real-time executions of a business activity can be estimated linearly by its real-time working time, that is demonstrated as follows:

$$WT_A = WT_{A_{pd}} * N_{td}/N_{pd} \tag{1}$$

WT_A denotes the estimated total working time of the business activity A with partial data, $WT_{A_{pd}}$ denotes the real-time working time of A when partial data is detected, N_{td} denotes the amount of the expected *output_data*, and N_{pd} denotes the amount of the partial data.

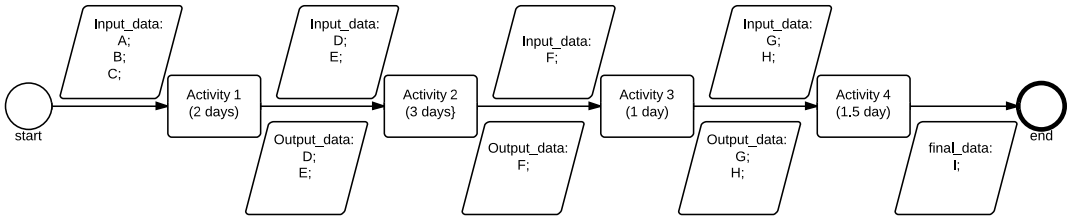
The partial data pattern for a single data item could be more complicated when the completion of the execution of A demands more than one re-execution of A. In this case, WT_A and N_{pd} are accumulated each time when A is re-executed and thus (1) can be modified as follows:

$$WT_A = \sum_{1}^{n}(WT_{A_{pd}}) * N_{td}/\sum_{1}^{n}(N_{pd}) \tag{2}$$

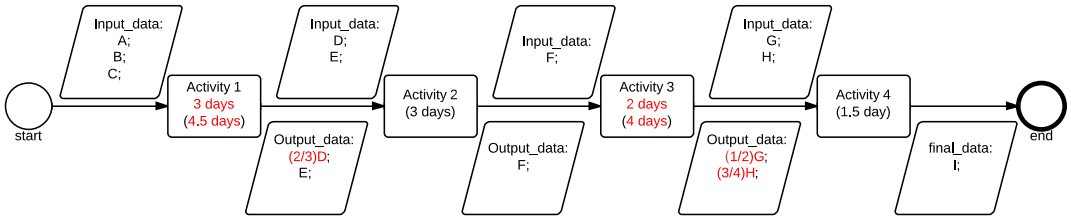
It is possible that partial data occurs to more than one data item during one execution of A. Since working time is a "worst-case" metric [9], to estimate WT_A, we simply estimate WT_A in terms of each data item and select the largest value among all possible values. The calculation is stated as follows:

$$WT_A = max_i(\sum_{1}^{n}(WT_{A_{pd}}) * N_{td_i}/\sum_{1}^{n}(N_{pd_i})) \tag{3}$$

Fig.1 illustrates the concepts of partial data by giving an example where Fig.1(a) is the predetermined business process with the apriori knowledge of working time for all the business activities, and Fig.1(b) indicates the runtime business process with the time performance caused by partial data highlighted. Activity 1 is presumed to be working for 2 days while in runtime, exceptions occur such that only 2/3 of data D is produced after 3 days. By applying (1), the estimated working time of activity 1 in this associated run time is calculated as 3/2 * 3 = 4.5 days. Similarly, the execution of activity 3 is delayed from the



(a) The predetermined business process



(b) The instantaneous execution of business process

Fig. 1. An example of partial data pattern

presumed 1 day to 2 days by producing 1/2 of data G and 3/4 of data H. The estimated working time of activity 3 due to data G and H is calculated as 4 days and 2.67 days. 4 days is selected as the estimated working time of activity 3 according to (3).

When partial *output_data* is produced, it might be able to be delivered for the execution of the subsequent business activities without waiting for the complete *output_data*. Fig.2 gives an example that when the subsequent business activities can be executed with partial *input_data* delivered from the execution of previous business activities. When activity 2 is allowed to use partial *input_data* (2/3*D) for its execution, activity 1', at mean time, is being executed to complete the *output_data* delivery. Since the execution of activity 1' takes 1.5 days, the subsequent execution of activity 2' is parallel to the execution of activity 2 (by that time, it still needs 0.5 day to complete its associated execution). In another sense, these parallel executions of business activities enable the combination of the execution of the two business activities into one, that is indicated by activity 2". On the other hand, if activity 2 is not executed immediately after the partial data D is input, while waiting to be executed after complete data D, the total working time of the subprocesses would be 1.5 days more. This example demonstrates the advantages of "execution whenever ready".

Note that in the pattern of partial data, all data types (i.e., *input_data*, *consumed_data*, etc.) are assumed to be quantified. Therefore, we do not need to

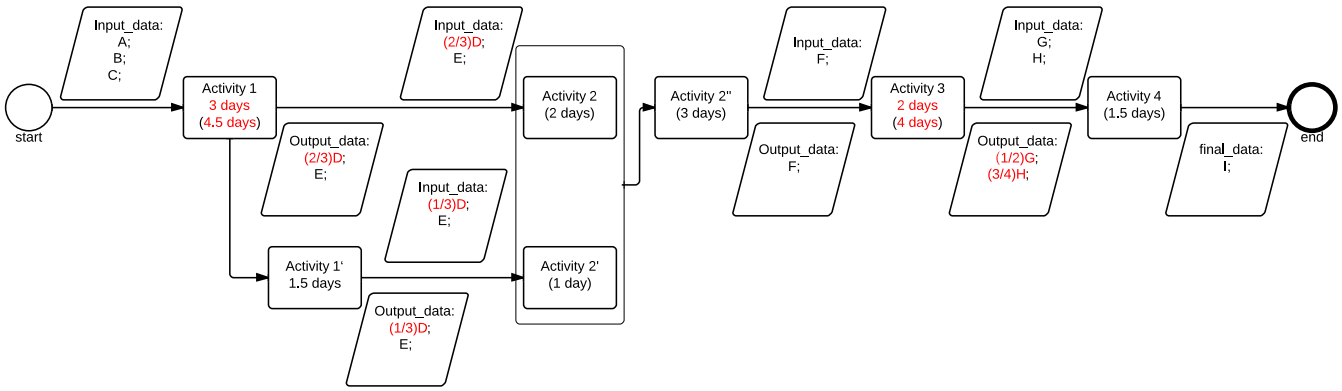


Fig. 2. An example of partial *output_data* delivery

consider the case that *input_data* is consumed already and cannot be delivered to the re-execution of the business activities with partial data, etc.

Halted Data The pattern of halted data is defined to represent the scenario in which the execution of any business activity is halted and cannot deliver any data, that is indicated by the more real-time working time than expected. Different from the pattern of partial data that aims at estimating the overall working time by obtaining the working time after the execution of business activities, the pattern of halted data is used to illustrate the overtime execution of business activities during runtime.

To better demonstrate the pattern of halted data, after being detected with more working time than expected, the working time of the associated executing business activity is updated at a certain frequency. When the execution of the business activity is eventually completed, the working time of that execution is obtained and the pattern of halted data is transformed to the pattern of partial data. On the other hand, after a certain predefined number of updates, if the execution of the associated running business activity has not been completed yet, the working time of this business activity is updated to be "∞", that indicates this business activity has encountered an exception that hinders the completion of its execution. Moreover, since the working time of this business activity is set to be "∞", the working time of the subprocess that this business activity is associated with is also set to be "∞".

Note that when the working time of the executing business activity with halted data is kept at "∞", it means that the subsequent business activities in the same subprocess usually cannot be executed and thus the execution of the associated subprocess is perceived to be failed (the approach with "best efforts" to "save" it is introduced in later sections). The frequency to update the working

time for the business activities with halted data is determined due to reality and not our concern in this paper.

Business Activity Insertion/Deletion In reality, business processes need to be changed by either inserting or deleting business activities for multiple reasons, such as changing business goals, varying runtime environments, and etc. Note that the patterns of changing business activities can be diverse [15]. However their impacts on the time performance of business processes are essentially identical that the time performance of subprocesses either increase or decrease with varying amount of business activities in business processes. In our approach, we represent all the patterns of changing activities by the typical business activity insertion/deletion to analyze their impacts on the time performance.

Generally when new business activities need to be inserted or deleted, they are intentionally designed with proper data types such that their insertions/deletions can ensure the delivery of the updated data flows. In our approach, it is assumed that these business activities are well designed and no errors of execution by the designs of them need to be considered.

4.2 Algorithm of Time Optimization

Our algorithm is initiated by detecting the exception patterns and followed with the algorithm for each exception pattern. This is equivalent to "divide and conquer" and advanced when multiple types of exceptions occur at the same time. This is because the defined exception patterns in our approach are orthogonal to each other, that is, their formulations and properties do not necessarily involve each other, which provides the orthogonality for their solutions and enables their additivity in multi-exception scenarios.

Algorithms for the Pattern of Partial Data In the pattern of partial data, the runtime working time is deviated from the predetermined working time of the associated business activity to render the time optimality of the original business process design incorrect. Approaches of statically optimizing time performance of business processes by relocating the execution order of business activities are introduced in our previous research [9]. Our approach in this paper, based on [9], applies the techniques of business activity relocation to optimized time performance for runtime business processes.

The relocation of a business activity A needs to ensure that the data flows cannot be hindered in both the subprocess that A is originally located in, namely $S_{original}$ and the one that it is relocated to, namely $S_{relocate}$. Intuitively, without the delivered data from A in $S_{original}$ and with the data that can be delivered to A in $S_{relocate}$, the fact that the data flows of both subprocesses are successfully delivered can validate the correctness of the relocation of A. This intuition provides the basis of the eligibility for business activity relocation.

Moreover, business activity relocation in runtime business processes is not allowed to trespass the subprocesses that have already been executed to avoid wastes of resources. That requires a precise understanding of the executing

business activities that distinguish the subprocesses that have not been executed from the the ones that already have. Accordingly, a cut $C_{executing} = (A_1, A_2, ..., A_i)$ is defined as the business activities A_is that are executing at the same time. The techniques to discover the business activities that are being executed at the same time have been well studied [14] thus are not our concern in our approach.

$C_{executing}$ indicates the starting points of the business process during runtime, that further indicates that the business activities that have not been executed should not be relocated before $C_{executing}$.

To help identify the eligibility of business activity relocation during runtime, for a certain business activity A, a data set $D_{input\_reserve}$ is defined to store the data that are delivered yet not consumed from all the subprocesses before the execution of $S_{relocate}$. Note that in our approach of this paper, the amount of data needs to be taken into the consideration of data types. $Input_{beforeS_{relocate}}$, $output_{beforeS_{relocate}}$, $Initial_{beforeS_{relocate}}$, and $consumed_{beforeS_{relocate}}$ respectively denote the sets of all the *input_data*, *output_data*, *initial_data*, and *consumed_data* of the subprocesses that are executed before $S_{relocate}$.

Definition 1 $D_{input\_reserve} = output_{beforeS_{relocate}} + consumed_{beforeS_{relocate}} - (input_{beforeS_{relocate}} + initial_{beforeS_{relocate}}) \cap (output_{beforeS_{relocate}} + consumed_{beforeS_{relocate}})$

Similar to [9], the eligibility of a business activity being backwardly movable under the circumstance of partial data is as follows:

Theorem 1 *A business activity* A *is* ***backwardly movable*** *to a subprocess* S *(*A *and* S *are executed no earlier than* $C_{executing}$*) when* A *is executed after* S*, and its set of input_data* $input_A$ *only belongs to* $D_{input\_reserve}$.

As opposed to $D_{input\_reserve}$, a data set $D_{output\_reserve}$ is defined to store the data that is delivered to all the subprocesses after the execution of $S_{relocate}$.

Definition 2 $D_{output\_reserve} = input_{afterS_{relocate}} + initial_{afterS_{relocate}} - (input_{afterS_{relocate}} + initial_{afterS_{relocate}}) \cap (output_{afterS_{relocate}} + consumed_{afterS_{relocate}})$

The eligibility of the relocation of business activities being forwardly movable and loosely-paralleled movable are defined as follows:

Theorem 2 *A business activity* A *is* ***forwardly movable*** *to a subprocess* S *(*A *and* S *are executed no earlier than* $C_{executing}$*) when* A *is executed before* S*, and its set of output_data* $output_A$ *only belongs to* $D_{output\_reserve}$

Theorem 3 *A business activity* A *is* ***loosely-paralleled movable*** *to a subprocess* S *(*A *and* S *are executed no earlier than* $C_{executing}$*) when* A *is loosely parallel to* S*, and* output_A *only belongs to* $D_{output\_reserve}$ *and* input_A *only belongs to* $D_{input\_reserve}$.

The details about the execution order between $S_{relocate}$ and A (i.e., executed before, executed after, and loosely parallel) are introduced in our previous approach [9]. With the eligibility of business activity relocation, and the boundary of the runtime business processes defined by $C_{executing}$, the following steps to optimize the time performance of the overall business processes in the pattern of partial data is equivalent to the static time optimization approach in [9]. Note that the eligibility of business activity relocation is also the prerequisite of the algorithms to optimize time for the patterns of halted data and business activity insertion/deletion.

Algorithms for the Pattern of Halted Data A subprocess with halted data hinders not only its execution, but also possibly the execution of the overall business process when it bounds under convergence relations, such as "join" of BPMN models [16]. Similar to the pattern of partial data, our approach for halted data pattern is initiated by detecting $\mathrm{C}_{executing}$ when the working time of the executing business activity with halted data turns to "∞".

Define D_{input_cut} as the set of *input_data* of all the subprocesses that have been executed before the instantaneous cut of the executing business activities. Here in our approach at this point, data is assumed to be easily replicated, i.e., *consumed_data* is not considered. Therefore D_{input_cut} is defined as $input_{cut} \cup initial_{cut} \cup output_{cut}$, where $input_{cut}$ and $initial_{cut}$, and $output_{cut}$ respectively represent *input_data*, *initial_data*, and *output_data* of the subprocesses that have been executed before $C_{executing}$.

With the completion of the execution of more subprocesses, D_{input_cut} is expected to include more data items. It is possible these data items include *input_data* for business activities that await being executed in the subprocess with the halted data. If D_{input_cut} happens to include all the *input_data* to execute any of these business activities, then this subprocess is possible to be executed by "skipping" the execution of the business activity with halted data.

For each business activity A_i that has not been executed in the subprocess with halted data pattern, its *input_data* set $input_i$ is recorded. Each time when the execution of any activity in the business process is completed, $C_{executing}$ and D_{input_cut} are synchronously updated. Then for each of A_i, its $input_i$ is checked against D_{input_cut}. If any $input_i \subset D_{input\_cut}$, A_i is qualified to be executed directly instead of waiting for the completion of the execution of the one with the halted data.

Note that it is possible that there might be some business activities that cannot be executed between the one with the halted data and the executing one in the same subprocess. According to our business process modeling approach [8], the data which is supposed to be produced by their executions might be *input_data* to trigger the execution of later executed subprocesses. Thereby, it is possible that the execution of the business process still fails to proceed. On the other hand, this possibility validates that our approach offers "best efforts".

Given more than one subprocess that suffer from the pattern of halted data, the solution above is simply repeated for each of these subprocesses.

Business Activity Insertion/Deletion Assuming the business process that is originally designed has an optimized time performance already for simplicity, our approach to solve business activity insertion is initiated with a simple scenario where only one business activity needs to be inserted to an runtime business process. Since it is also assumed that the design of the inserted business activity is correct, there must be at least one possible placement for this business activity to be inserted in a certain subprocess, and each of the subprocesses with the possible placements is collected.

To insert a business activity demands an understanding of the time performance of each subprocess with the possible placements, namely S_i of the runtime business process. Define their working time as WT_{S_i}, and the working time of the cluster (defined as subprocesses that are bounded by the identical convergence or divergence relations) where S_i belongs to, namely $WT_{S_i\_cluster}$ is defined as the largest working time of its subprocesses. $Dif_{S_i\_cluster}$ is defined and calculated as $WT_{S_i\_cluster}$ - WT_{S_i}. If $Dif_{A\_S_i}$ (defined as $Dif_{S_i\_cluster}$ - WT_A) is non-negative, it indicates the $WT_{S_i\_cluster}$ cannot be changed by inserting the business activity A into this subprocess, and the time performance of the overall business process can be kept the same optimized as designed. If there are multiple such placements for the business activity insertion, it is as simple as to pick the one with the largest $Dif_{A\_S_i}$ for the insertion. On the other hand, if $Dif_{A\_S_i}$ is negative for all its possible placements of the business process, the business activity A is inserted into the subprocess that makes the largest $Dif_{A\_S_i}$ to approach an optimized time performance of the overall business process.

Time optimization of the overall business process is expected to be harder to approach when there are multiple business activities needed to be inserted into a runtime business process. A straightforward method is to randomly insert all of the business activities into the proper subprocesses and apply the static time optimization approach of [9] to realize an optimized time performance. However, as discussed in [9], the time complexity is expected to be high. In our approach in this paper, with the apriori knowledge of the possible placements of the inserted business activities, the freedom of degree of the business activity insertions in subprocesses is expected be lower than the approach of [9]. Accordingly, the time complexity of business activity intertion is lower. Typically in reality, the possible placements of the business activity insertions can be easily captured and stored in an array *[A]* where the activities are sorted according to the number of their possible inserted subprocesses in an ascending order. A_i with fewer possible placements are inserted earlier into the business process. If it has more than one possible placement, it would be applied with the approach of inserting one business activity to ensure the optimized time performance. This process repeats until every business activity is inserted to the business process.

Intuitively, the methods above can approach an optimized time performance of the overall business process. However, that is not always true. For instance in Fig.3, assume A_i can be inserted to two subprocesses S_1 and S_2 to maintain an optimized time performance (10.5 days). According to our approach to insert only one business activity, A_i is inserted to S_1. Assume A_{i+1} can be inserted to S_1, S_2, and S_3. Any insertion at this point would increase the working time of

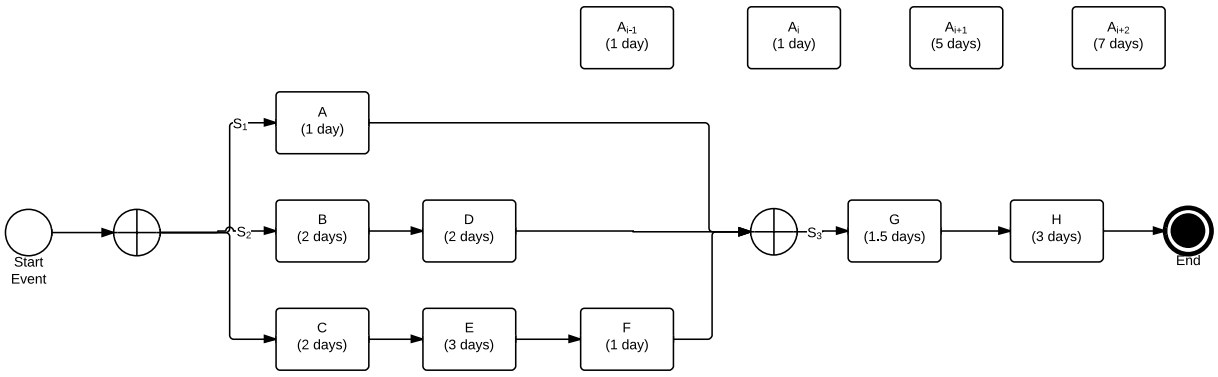


Fig. 3. An example of "backtracking" process for business activity insertion

the overall business process while the insertion to S_1 would increase the least among them (11.5 days). Therefore A_{i+1} is inserted to S_1. It is clear that if A_i could be inserted to S_2 and A_{i+1} could be inserted to S_1, the time performance of the overall business process can be better improved (10.5 days). To avoid this, an extra "backtracking" process is added into our approach: each time a business activity A_{i+1} from *[A]* is inserted to the business process and increases the working time of the overall business process, the previous placement of A_{i-j} (if there is any) in the same subprocess is checked for whether it can be reinserted to other subprocesses to reduce the working time of the overall business process. If so, A_{i-j} is reinserted and a better time performance of the overall business process is achieved.

This "backtracking" process, though, cannot completely eliminate similar counter examples as the one above. For instance, assume A_{i-1} and A_i are inserted to S_1 to maintain an optimized time performance of the overall business process. The assumption that A_{i+2} can only be inserted to S_1 to ensure the smallest increase of the working time of the overall business process might demand "backtracking" the possible reinsertions of both A_{i-1} and A_i to approach a better time performance. This apparently increases the time complexity of the algorithm. Similar as [9], we make trade-offs with efficiency for our approach of this paper by performing the "backtracking" process within limited times.

The methods to realize optimized time performance of the overall business process under the pattern of business activity deletion is as simple as to apply the approach of static time optimization of [9] for the overall business process, that is bounded by the cut $C_{executing}$ with business activities deleted.

The pseudo code of the the algorithms for all the exception patterns to optimize time performance of business process while handling all the exception patterns is listed as follows:

Algorithm 1 Dynamically_Optimize_Time (Business_Process): dynamically deriving the optimized time for the overall business process when handling runtime exceptions

Input:
BP := business process that needs to optimize time performance when handling runtime exceptions,
EP := exception patterns, including partial_data, halted_data, insertion, deletion
A := business activitiy
S := subprocess
[A] := array of business activities that need to be inserted to BP in an ascending order of the number of possible placements
$C_{executing}$:= (A_1, A_2, ... A_i), a cut to indicate the running business activities in the business process
A_Δ := new business activity being part of $C_{executing}$
Exception_Flag := flag to indicate the exception detected
D_{input_cut} := input_data set of all the subprocesses that have been executed before $C_{executing}$
A_{halted} := business activity in the subprocess with halted data that have not been executed
$input_A$:= the input_data set of a business activity
WT_A := the working time of a business activity
$WT_{S_i\_cluster}$:= the working time of the cluster in which S_i is associated with
$Dif_{S_i\_cluster}$:= varying working time between S_i and its associated cluster
$Dif_{A\_S_i}$:= varying working time between $Dif_{S_i\_cluster}$ and the working time of A
i,j,k,m := integer
Output:
$Optimize\_Time(BP)$:= the optimized working time of the overall business process

if *Exception_Flag* **then**
  if EP := partial_data **then**
    update the working time for business activities with partial data;
    reconfigure BP bounded by $C_{executing}$;
    Statically_Optimize_Time (BP);
  end if
  if EP := halted_data **then**
    if $C_{executing}$:= (A_1, A_2, ... $A_{i+\Delta}$) **then**
      update D_{input_cut};
      for each $A \in \{A_{halted}\}$ **do**
        if $input_A \in D_{input\_cut}$ **then**
          execute A;
          update $C_{executing}$ and D_{input_cut};
          break;
        end if
      end for
    end if
  end if
  if EP := deletion **then**
    delete business activities;
    reconfigure BP bounded by $C_{executing}$;
    Statically_Optimize_Time (BP);
  end if
  if EP := insertion **then**
    while i < sizeof(*[A]*) **do**
      for each S that A_i can be inserted to **do**
        if $Dif_{A_i\_S} >= 0$ **then**
          insert A_i to S with the largest $Dif_{A_i\_S}$;
          i := i + 1;
          break;
        end if
        if $Dif_{A_i\_S} < 0$ **then**
          continue;
          check whether A_{i-m} is in any S;
          if A_{i-m} is in S_k and can be inserted to S_j to reduce the most of time **then**
            update $WT_{S_k\_cluster}$ and $WT_{S_j\_cluster}$;
            insert A_i to the subprocess with the smallest $WT_{S_k\_cluster} + WT_{S_j\_cluster}$;
            if this subprocess is S_k **then**
              insert A_{i-m} to S_j;
            end if
          end if
          insert A_i to S with the largest $Dif_{A_i\_S}$;
          i := i + 1;
          break;
        end if
      end for
    end while
  end if
  update $Optimize\_Time(BP)$;
end if

5 Conclusion and Future Work

In this paper we study the approach to dynamically optimize time performance of business processes when handling runtime exceptions. The contributions of our approach are: *1)* we propose the exception patterns that are possible to be handled in runtime business processes, i.e., partial data, halted data, and business activity insertion and deletion, in a data-centric manner, and further explore their impacts on the time performance of business processes. *2)* for each of the exception patterns, algorithms are developed to dynamically handle them during runtime to approach optimal time performance for the overall business process.

For the pattern of partial data, the working time of the executing business activity with partial data is re-estimated by its runtime performance. To approach the optimal time performance, the dynamic reconstruction of the original business process is then equivalent to our previous approach of statically optimizing time performance [9]. For the pattern of halted data, it is possible that, instead of waiting for being chronologically executed after the completion of the execution of the business activity with halted data, the subsequent business activities can be directly executed when data delivered by the execution of other subprocesses happen to include all of their input_data. Therefore, "best efforts" are provided to proceed executing the business activities in the same subprocess with the halted data. For the pattern of business activity insertion, business activities are inserted to the subprocesses that can enable an optimal time performance of the overall business process. A "backtracking" process is applied when two business activities are inserted in the same subprocess to inspect whether they contribute to the time optimization. The methods to realize time optimization of the exception pattern of business activity deletion is essentially that of statically optimizing time performance after deleting unnecessary business activities from business processes.

To verify our approach in the future, we plan to simulate these methods to find out their efficacy and improve these methods due to the feedback from the evaluation results. We also plan to conduct case studies for industries to gain the feedback from the business actors and enterprise software system developers for the future improvements.

Time is one of the major factors that need to be seriously considered and optimized when constructing a business process. Other major factors include financial costs, human labor involvement, etc. Dynamic exception handling techniques could be valuable when they can improve these major factors of business process modeling. Moreover, the research of this technique associated with specific scenarios (e.g, product line, etc.) can be expected to more useful than a holistic overview or theory.

References

1. F. Leymann and D. Roller. Business process management with flowmark. In *Compcon Spring '94, Digest of Papers.*, pages 230–234, Feb 1994.

2. J. Eder, E. Panagos, H. Pezewaunig, and M. Rabinovich. Time Management in Workflow Systems. In W. Abramowicz and M.E. Orlowska, editors, *Third International Conference on Business Information Systems (BIS'99)*, pages 265–280, Poznan, Polen, 1999. Springer-Verlag, Berlin.
3. Yang Yu, Ting Xie, and Xiaoyan Wang. A handling algorithm for workflow time exception based on history logs. *The Journal of Supercomputing*, 63(1):89–106, 2013.
4. Wangyang Yu and Xianwen Fang. Analyzing real-time predictability of business processes based on petri nets. In Zhixiang Yin, Linqiang Pan, and Xianwen Fang, editors, *BIC-TA*, volume 212 of *Advances in Intelligent Systems and Computing*, pages 207–215. Springer, 2013.
5. Xiaobo Guo, Jidong Ge, Yu Zhou, Haiyang Hu, Feng Yao, Chuanyi Li, and Hao Hu. Dynamically predicting the deadlines in time-constrained workflows. In Zhisheng Huang, Chengfei Liu, Jing He, and Guangyan Huang, editors, *Web Information Systems Engineering C WISE 2013 Workshops*, volume 8182 of *Lecture Notes in Computer Science*, pages 120–132. Springer Berlin Heidelberg, 2014.
6. Andreas Lanz, Roberto Posenato, Carlo Combi, and Manfred Reichert. Controllability of time-aware processes at run time. In Robert Meersman, Herv Panetto, Tharam S. Dillon, Johann Eder, Zohra Bellahsene, Norbert Ritter, Pieter De Leenheer, and Deijing Dou, editors, *OTM Conferences*, volume 8185 of *Lecture Notes in Computer Science*, pages 39–56. Springer, 2013.
7. K. Vergidis, A. Tiwari, and B. Majeed. Business process analysis and optimization: Beyond reengineering. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(1):69–82, Jan 2008.
8. Yuqun Zhang and Dewayne E. Perry. A goal-directed method towards business process modeling. In *Service-Oriented System Engineering, 2014. IEEE International Conference on*, volume 12, Apr 2014.
9. Yuqun Zhang and Dewayne E. Perry. A data-centric approach to optimize time in workflow-based business process. In *Services Computing, 2014. IEEE International Conference on (to appear)*, volume 8, Apr 2014.
10. Dewayne E. Perry, Nancy A. Staudenmayer, and Lawrence. Understanding and improving time usage in software development. In Alfonso Fuggetta and Alexander L. Wolf, editors, *Trends in Software Process*, chapter 5. John Wiley and Sons, 1996.
11. Dewayne E. Perry, Nancy A. Staudenmayer, and Lawrence G. Votta. Understanding software development: Processes, organisations and technologies. *IEEE software*, 11:36–45, 1994.
12. Hafedh Mili, Guitta Bou Jaoude, Eric Lefebvre, Guy Tremblay, and Alex Petrenko. Business process modeling languages: Sorting through the alphabet soup. In *OF 22 NO. IST-FP6-508794 (PROTOCURE II) SEPTEMBER*, page 2005, 2004.
13. George M. Giaglis. A taxonomy of business process modeling and information systems modeling techniques. *International Journal of Flexible Manufacturing Systems*, 13(2):209–228, apr 2001.
14. Yutian Sun and Jianwen Su. Computing degree of parallelism for bpmn processes. In *Proceedings of the 9th international conference on Service-Oriented Computing*, ICSOC'11, pages 1–15, Berlin, Heidelberg, 2011. Springer-Verlag.
15. Han Aa, HajoA. Reijers, and Irene Vanderfeesten. Composing workflow activities on the basis of data-flow structures. In Florian Daniel, Jianmin Wang, and Barbara Weber, editors, *Business Process Management*, volume 8094 of *Lecture Notes in Computer Science*, pages 275–282. Springer Berlin Heidelberg, 2013.
16. Object Management Group (OMG). Business process model and notation (bpmn) version 2.0. Technical report, jan 2011.