A Data-Centric Approach to Optimize Time in Workflow-Based Business Process

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*—Business process optimization can provide direct benefits to achieve business goals by improving predetermined objectives. In this paper, we study the properties of time performance and develop approaches to optimize it in workflow-based business processes. By applying our data-centric business process modeling techniques, we explore the possibility of reconstructing business process that is realized by modifying the execution order of business activities. Accordingly, we develop efficient algorithms to approach optimal time performance for both a single subprocess and an overall business process.**

***Keywords*-time performance, data-centric, business process optimization**

I. Introduction

In the realm of business process modeling, optimization refers to the automated improvement of business processes using pre-specified quantitative measures of performance (objectives) [1]. A holistic approach for business process modeling should optimize business processes and eventually generate improved ones [1] [2]. However, in the past decades, while most of research approaches about business process modeling have been proposed to cover the modeling techniques and qualitative analysis, business process optimization, e.g. time optimization, has gained less attention.

One possible reason is that some researchers perceive time as only a carrier of resources. They emphasize that business process optimization should reduce time and cost to improve the specified product qualities [3] [4]. It is nevertheless argued that this multi-aspect optimization might be complicated and result in conflicting criteria [5]. A large portion of the relevant research is based on scheduling, that mathematically models the optimal resource allocation of business processes in terms of specified objectives [6] [7] [8] [9]. While these approaches are considered to optimize time-relevant business applications, it is argued in [10] that they can only be applied in simplified and ideal business processes where their constraints and objectives can be mathematically modeled. However, some time-relevant business elements, such as decisions, are hardly mathematically expressed. Moreover, modeling techniques for these elements might not be applied when their constraints vary in realistic business process settings.

On the other hand, a group of researchers insist that time is essentially a significant resource of process management. Time is considered as one important factor in modeling and evaluating business processes [11] [12]. Some time-relevant properties, such as the degree of parallelism [13], the race time and elapsed time [14], etc., have been widely explored. Though being specific in depicting these time-relevant properties, they mainly focus on the conclusions of their discoveries from empirical studies or domain-specific interests, instead of providing explicit methodologies about how to exactly improve these properties.

In this paper we study an approach to optimize time performance of workflow-based business process, that is based on our previous research of data-centric business process modeling techniques [15]. In a workflow-based business process, a business activity is defined to consist of atomic activities, data types, and human actors. Moreover, we use an abstract syntax tree (*AST*) to represent a business process. Assuming deterministic components of business activities that cannot be redesigned in our approach at this point, our time-optimization approach aims at reconstructing business processes by modifying the execution order of business activities, i.e., by relocating business activities from their original subprocesses to others, where a subprocess is defined to be composed of business activities in execution order and bounded by their convergence and divergence relations.

Assuming an apriori knowledge (i.e., either known or estimated) of time of each business activity as [13] [16], our approach is initiated by determining whether a business activity is eligible for relocation in a business process. Given a subprocess that a business activity is not originally located in, the business activity is determined to be eligible for relocation to the subprocess only when it is, as defined in our approach, forwardly moveable, backwardly moveable, or loosely-parallelled moveable to the subprocess.

For any subprocess, given the working time that is defined as the total lapse time to complete a process unit (e.g., business activity, subprocess, and process), as in [14] [17], its waiting time is defined as the time duration of its idle state when other subprocesses are being executed. In our approach, the time optimization of a subprocess is essentially to optimize the waiting time by filling its idle states with executing eligible business activities from other subprocesses. By selecting the proper eligible activities according to their contributions to the waiting time performance of the subprocess, our algorithm effectively reconfigures the subprocess by inserting those activities into it to minimize its waiting time.

Furthermore, we develop algorithms to optimize time performance for the overall business process. Based on the technique of optimizing time performance for a single subprocess, the time-optimization algorithm of the overall business process modifies the sorting criteria of eligible activities to emphasize their contributions to the overall time optimization of the business process, and realizes the efficient arrangement of the business activity relocation for the subprocesses that need to optimize the time performance.

The rest of the paper is organized as follows. Section II introduces related work. Section III revisits our abstract-syntax-tree-based technique and its associated properties. Section IV describes our approach of optimizing time performance of business processes. Section VI discusses our conclusions and future work.

II. Related Work

Scheduling is one major approach to optimize time-relevant properties in business processes. Researchers of [6] present an overview of mixed integer linear programming (MILP) based approaches to schedule chemical processing systems. By representing time in a discrete and continuous manner, they develop effective models for a variety of chemical processes and efficient solutions for difficult MILP models in a short-term scheduling domain. Using the similar time representation, researchers of [8] present two scheduling models: single-unit assignment models where task assignment is predetermined and multiple-unit assignment models where objects compete for processing products. Researchers of [9] aim at extending scheduling techniques of batch processes to handle large volume processes as well as different objectives. However, they are argued to be limitedly applied in the scenarios in which constraints cannot be mathematically modeled. Moreover, most of the techniques are NP-complete such that closed-form solutions cannot be provided by merely deriving those mathematical models.

Time sometimes is considered as a reference for modeling techniques. Researchers of [18] [12] include time as the dynamic view along with informational, functional, and organizational views together as the fundamental views to construct business processes. Assuming apriori knowledge of working time of each business activity, researchers of [13] present methods to detect the degree of parallelism, that is defined as the number of the running subprocesses at the same time. Researchers of [14] [17] implement empirical studies of software developments and try to identify the factors to cause the discrepancy of race time, that is defined as the time spent on actual work and elapsed time, that additionally includes the presence of interruption, blocking, and waiting periods. However, these researches fail to provide in-depth approaches for how exactly time performance can be improved by applying their discoveries. Our approach, by analyzing the eligibility of how a business process can be optimized, provides methods to effectively optimize time performance for both a single subprocess and overall business process.

III. *AST-based* Technique Revisited

In our previous research, a workflow-based business process is represented in an abstract syntax tree (*AST*) with a specified set of syntax [15]. In the *AST-based* approaches, a business activity being composed of a tuple of components (data, human actor, and atomic activity) is formed when one or more atomic activities are executed in order by identical human actor(s). 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 initially injected to the data flows and triggers the execution of the associated business activity.
- *input_data*: data that is generated and delivered by 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 by the execution of the associated business activity.
- *output_data*: data that is the deliverable by the execution of the associated business activity.
- *final_data*: data that is generated 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.

In our approach, we do not consider *global_data* for its contribution to evaluate any property of a business process because it is a constant.

A business process is bounded by the divergence and convergence relations of “cooperative”, “exclusive”, “N-cooperative”, and “N-exclusive”. The former two concepts are identical with what they are in [19]. “N-cooperative” indicates the number of N out of M business activities (N<=M) or more are allowed to be executed to proceed to the subsequent executions. “N-exclusive” indicates only the number of N out of M business activities (N<=M) or less are allowed to be executed to proceed to the subsequent executions.

An example *AST-based* business process is shown in Fig.1, with the letters representing business activities. It depicts a business process that is initiated by the execution of F, H, and J. G is executed after H. E is executed after the execution of either F or G. D, that is executed after

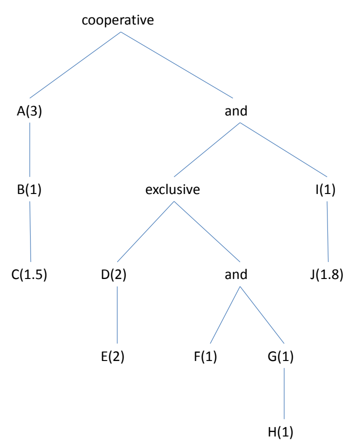

Figure 1. An AST example

E, together with I, that is executed after J, are executed to trigger the subsequent execution of C, B, and A. The *AST-based* technique renders the following properties:

- Operators (cooperative, exclusive, N-cooperative, and N-exclusive) are placed as non-leaf nodes. Leaf nodes must be business activities.
- Business activities are placed left when its parent node is an operator (i.e., cooperative, exclusive, N-cooperative, or N-exclusive). Business activities that are executed to trigger the execution of the subsequent subprocess, are placed under "and" following the operators.
- Leaf nodes that are placed in lower levels are executed before those placed in higher levels.
- Operators are associated with the connector "and" as one of their child nodes.

Our *AST-based* approach provides a goal-directed modeling technique of business process, where business goals are modeled as final states of data. Starting from these final states, metrics (e.g. data dependencies) are evaluated by backtracking the business process through the *AST*. Moreover, the definition of the data types and their transmission patterns enable a data-centric approach where business process is essentially realized, maintained, and depicted by data flows. Compared with traditional business process modeling techniques, e.g. BPMN [19], our *AST-based* technique is advanced in that *1)* it breaks acyclic graphs into regular tree structures; *2)* it provides more complicated convergence and divergence relations (i.e., "N-cooperative" and "N-exclusive"); *3)* it provides possible fast and easy solutions by using sophisticated tree-relevant techniques.

Our research in this paper, based on the *AST-based* approach, analyzes the scenarios where time performance of business process can be optimized, and in turn, provides the methodologies to optimize time for both single subprocess and overall business process.

IV. Approach Overview

In this section, we introduce our approach to optimize time performance of business processes, that is classified to two stages: the detection of the eligibility for business activity relocation and the algorithms of time optimization. Typically, our approach is initiated by identifying whether one business activity is eligible to be relocated to a given subprocess. With the knowledge of the eligibility and the working time of each business activity, our algorithm selects proper eligible activities for the subprocess to approach an optimal time performance. Accordingly, the time optimization of the overall business process is approached by modifying the criteria of selecting the eligible activities and effectively assigning them to the subprocesses that demand time optimization.

A. Eligibility for Activity Relocation

In a workflow-based business process, a business activity is executed in a chronologically static manner when its input is delivered by its adjacently previous activities, or its output is immediately absorbed by its adjacently subsequent activities. On the other hand, it is possible that a business activity can be flexibly executed if it is not directly dependent on the execution of its adjacently- previous or subsequent activities.

Our *AST-based* approach specifies this possibility by defining that for any execution of adjacent business activities, the *input_data* of the later-executed activity is not necessarily the *output_data* of the former-executed activity. Specifically, assume a business activity, namely *BA*, with its *input_data* being part of the *output_data* generated only by a subprocess SP_A that is executed earlier but not adjacently ahead. Then the advanced execution of *BA* adjacently following SP_A is legitimate (assuming no other sources of impact), because this flexible execution does not impact on their original order of data transmission that ensures the execution of the business process. Analogously, if the *output_data* of *BA* is part of the *output_data* of a subprocess SP_B only and SP_B is executed not adjacently later, then the postponed execution of *BA* adjacently ahead of SP_B is legitimate as well. This observation provides the fundamentals for flexible execution of business activities in business process, that is realized in the *AST-based* approach as the relocation of the eligible business activities.

Our approach in detecting the eligibility for business activity relocation is initiated by representing a business process in *AST* and printing the paths from the root to all its leaf nodes, namely *P*. These paths are then sorted by their mutual commonalities and lengths of the operators and connectors (i.e., "cooperative", "exclusive", "and", etc.). For instance, $\{P\}$ in Fig.1 are originally represented in *AST* as {(*cooperative*, A, B, C), (*cooperative*, and, *exclusive*, D, E), (*cooperative*, and, *exclusive*, and, F), (*cooperative*, and, *exclusive*, and, G, H), (*cooperative*, and, I, J) }, where their paths that have been tailored to include only the operators, namely *TP*, are represented and synchronically sorted with

$\{P\}$ as $\{$(*cooperative*), (*cooperative*, and), (*cooperative*, and, *exclusive*), (*cooperative*, and, *exclusive*, and), (*cooperative*, and, *exclusive*, and)$\}$. It is observed that the adjacent sorted paths of operators differ at most by one operator element.

According to our *AST* properties, the execution order of business activities of the sorted $\{P\}$ can be reflected by the number of the operator elements ahead of them in their own paths. For instance, activity D and E of (*cooperative*, and, *exclusive*, D, E) are executed after the execution of activity F of (*cooperative*, and, *exclusive*, and, F) because both of the paths have the same operator elements except the latter has an extra connector "and" that indicates F is executed earlier than D and E and trigger their executions.

This reflection of the execution order of business activities provides a possibility of deriving a partial order of any given subprocess and activity. Define *TP(A)* to denote the array of operator elements ahead of activity A , and *TP(SP)* to denote the array of operator elements ahead of subprocess SP in a *TP*. The partial order between A and SP can be derived by comparing the elements in *TP(A)* and *TP(SP)*.

Theorem 1: Activity A is **executed before** subprocess SP when *TP(SP)* $\subset$ *TP(A)*, and Activity A is **executed after** subprocess SP when *TP(A)* $\subset$ *TP(SP)*.

A *TP* is captured from a root to a leaf node in *AST*. Therefore, any *TP* that is a subset of another indicates the corresponding activity/subprocess is in the same workflow with the other, and their execution order can be completely identified.

Theorem 2: Activity A is **loosely parallel** to subprocess SP when *TP(SP)* $\not\subset$ *TP(A)*, and *TP(A)* $\not\subset$ *TP(SP)*.

As opposed to Theorem 1, The fact that the *TP*s of a pair of activity/subprocess are not mutually subsets of one another indicates disjoint workflows between them. This parallelism is considered to be loose because the chronology of A and SP can actually be detected by deriving the total order of activities/subprocesses based on the properties of the *AST-based* approach and the knowledge of their working time, yet it is not necessary in our approach at this point.

In the data-centric *AST-based* approach, a workflow-based business process is perceived as being established and maintained by its dataflow, i.e., the execution of a business process is realized by the data delivery among business activities. Hence to explore the eligibility of relocating a business activity of a business process is essentially to find out whether the data delivery in the business process would be impeded after relocating the business activity. Assuming in our approach at this point that a business activity is kept consistent without modifying its components (i.e., data types, human actors, and atomic activities), data delivery is ensured to be safe when its pattern, specifically the order of data transmission among business activities, is kept identical with that in the original workflows. Particularly, for any relocated activity, the data delivery of a business process is secured when the *input_data* of that activity is part of the data that is delivered by all its former-executed activities (e.g., the *output_data* that has not been consumed), and the *output_data* of that activity is part of the data that is delivered to all its later-executed activities (e.g., the *input_data*) in its workflow.

For instance, in Fig. 1, assume the *input_data* of activity C, that is obviously the part of the *output_data* generated by subprocess (F, E, D), (H, G, E, D), and (J, I), happens to be the part of the *output_data* generated by subprocess (J, I) as well. If activity C is relocated under the connector "and" and above activity I, its associated pattern of data delivery would stay identical with its original pattern. Specifically, the *output_data* generated by activity C could still be delivered to the latter-executed activities in the original workflow. In this case, the relocation of activity C is perceived as being eligible. Note that although the working time of the updated (J, I, C) is increased to be 4.3, the total working time of the subprocess of (*cooperative, and,* *) remains at 6, and the total working time of the overall business process is reduced by 1.5 (The numbers beside activities denotes the working time of the associated activities.).

Definition 1: S_{target} denotes a subprocess. A_{target} denotes an activity that is not part of S_{target}. *Input*$_{A_{target}}$, *output*$_{A_{target}}$, *Initial*$_{A_{target}}$, and *consumed*$_{A_{target}}$ respectively denote the sets of all the *input_data*, *output_data*, *initial_data*, and *consumed_data* of A_{target}. $S_{beforetarget}$ denotes the set of the subprocesses that are **executed before** S_{target} (including S_{target}) in its workflow. *Input*$_{S_{beforetarget}}$, *output*$_{S_{beforetarget}}$, *initial*$_{S_{beforetarget}}$, and *consumed*$_{S_{beforetarget}}$ respectively denote the sets of the *input_data*, *output_data*, *initial_data*, and *consumed_data* of all the activities of $S_{beforetarget}$.

$S_{beforetarget}$ can be derived by searching in the $\{$*TP*$\}$ the *TP*s that contains *TP(*S_{target}*)*. This search can be simple since $\{$*TP*$\}$ have already been sorted by their mutual commonalities and lengths of the operators.

Theorem 3: A_{target} is **backwardly moveable** to S_{target} when A_{target} is **executed after** S_{target}, and *input*$_{A_{target}}$ only belongs to $\{$*output*$_{S_{beforetarget}}$ + *consumed*$_{S_{beforetarget}}$ - (*input*$_{S_{beforetarget}}$ + *initial*$_{S_{beforetarget}}$) $\cap$ (*output*$_{S_{beforetarget}}$ + *consumed*$_{S_{beforetarget}}$)$\}$.

Theorem 3 illustrates under what circumstances the execution of A_{target} is eligible to be brought forward to be adjacently following the execution of S_{target}. According to the *AST-based* approach, it is possible that in $S_{beforetarget}$, a single piece of data is transmitted among different activities as multiple data types. Therefore, *output*$_{S_{beforetarget}}$ + *consumed*$_{S_{beforetarget}}$ - (*input*$_{S_{beforetarget}}$ + *initial*$_{S_{beforetarget}}$) $\cap$ (*output*$_{S_{beforetarget}}$ + *consumed*$_{S_{beforetarget}}$) ensures accurate data delivered from $S_{beforetarget}$. Since A_{target} is brought forward, the impact of its *output_data* delivery to the subsequent subprocesses stay consistent and is not necessarily cared about.

Definition 2: $S_{aftertarget}$ denotes the set of the subprocesses that are **executed after** S_{target} (including S_{target}) in its workflow. *Input*$_{S_{aftertarget}}$, *output*$_{S_{aftertarget}}$, *initial*$_{S_{aftertarget}}$, and *consumed*$_{S_{aftertarget}}$ respectively denote the sets of the *input_data*, *output_data*, *initial_data*, and *consumed_data* of all the activities of $S_{aftertarget}$.

As opposed to the way of deriving $S_{beforetarget}$, $S_{aftertarget}$ can be derived by searching in the $\{TP\}$ the *TP*s that are contained *TP(S_{target})*.

Theorem 4: A_{target} is **forwardly moveable** to S_{target} when A_{target} is **executed before** S_{target}, and *output*$_{A_{target}}$ only belongs to $\{$*input*$_{S_{aftertarget}}$ + *initial*$_{S_{aftertarget}}$ - (*output*$_{S_{aftertarget}}$ + *consumed*$_{S_{aftertarget}}$) $\cap$ (*input*$_{S_{aftertarget}}$ + *initial*$_{S_{aftertarget}}$)$\}$.

Theorem 4 illustrates under what circumstances the execution of A_{target} is eligible to be postponed to the execution of S_{target}. Since A_{target} is postponed, the impact from its former-executed subprocesses to its *input_data* stay consistent and is not our concern.

Theorem 5: A_{target} is **loosely-parallelled moveable** to *TP(S_{target})* when A_{target} is **loosely parallel** to S_{target}, *output*$_{A_{target}}$ only belongs to $\{$*input*$_{S_{aftertarget}}$ + *initial*$_{S_{aftertarget}}$ - (*output*$_{S_{aftertarget}}$ + *consumed*$_{S_{aftertarget}}$) $\cap$ (*input*$_{S_{aftertarget}}$ + *initial*$_{S_{aftertarget}}$)$\}$, and *input*$_{A_{target}}$ only belongs to $\{$*output*$_{S_{beforetarget}}$ + *consumed*$_{S_{beforetarget}}$ - (*input*$_{S_{beforetarget}}$ + *initial*$_{S_{beforetarget}}$) $\cap$ (*output*$_{S_{beforetarget}}$ + *consumed*$_{S_{beforetarget}}$)$\}$. Moreover, the workflows where A_{target} is originally located should not be impeded.

Obviously, the conditions to trigger a **loosely-parallelled moveable** business activity are stricter compared with those of Theorem 3 and 4. It is expected that in reality, few business activities are expected to be **loosely-parallelled moveable** in a sophisticated business process design.

In sum, given a subprocess and an activity, the activity is eligible for relocation in the business process when it can be **forwardly moveable**, **backwardly moveable**, or **loosely-parallelled moveable** to a different subprocess.

B. Algorithms of Time Optimization

Definition 3: Subprocesses that are composed of activities and bounded by the identical convergence or divergence relations are defined as a "cluster", where one subprocess of a cluster is called a cluster process. In the *AST-based* approach, cluster subprocesses are represented following the connector "and".

In this paper, time inefficiency is considered to be caused only by the waiting time of cluster subprocesses, that is generated after their executions are completed while other cluster subprocesses are still being executed. Given the apriori knowledge of the working time of each business activity of business process, the working time of a cluster subprocess can be calculated by simply summing up the working time of all its activities. Then the waiting time of that cluster subprocess is derived by subtracting the largest working time of the cluster with its own working time. Note in our approach, working time and waiting time are considered as worst-case metrics. Particularly, the working time of a cluster is defined as the largest working time of all its cluster subprocesses, regardless the operator types. Therefore, even though operators "exclusive" and "N-exclusive" indicate the scenarios that only the subprocesses winning the competition are allowed to proceed to the subsequent execution of the business process, the working time of their associated cluster cannot simply be the shortest working time among all its cluster subprocesses. Moreover, the assumption of some business process modeling techniques [13] [15] that no exception of executing business activities does not always hold in reality and the subprocesses that have shorter working time cannot always be executed. For instance, in Fig. 1, the working time of subprocesses (F), (H, G), (F, E, D), (H, G, E, D), and (J, I) is respectively 1, 2, 5, 6, and 2.8. That leads to the waiting time of each calculated as 1, 0, 1, 0, and 3.2.

1) Time Optimization for A Single Cluster Subprocess: It is possible in reality that only one (or a few) subprocess needs to optimize its time performance to achieve certain business goals without considering the overall optimization for the entire cluster or business process. In our approach, to optimize time performance for a single cluster subprocess, namely S_{target} is simply equivalent to optimize its working time by filing itself with executing additional eligible activities from other subprocesses.

Each activity in the business process is detected beforehand whether they are eligible for S_{target}, that is, whether they are **forwardly moveable**, **backwardly moveable**, or **loosely-parallelled moveable** to S_{target}. There might be multiple eligible activities, namely $A_{eligible}$s in other subprocesses, all of which could contribute to optimizing waiting time of S_{target}. Ideally, they are considered to be selected and relocated together to S_{target} to minimize the working time. However, it is only allowed to have one $A_{eligible}$ relocated to S_{target} at one time, because after any activity is relocated to S_{target}, the patterns of data delivery of S_{target} need to be updated accordingly. It is possible that multiple $A_{eligible}$s might be "incompatible" with each other's workflow, such that data delivery of S_{target} would be impeded if they are relocated together to S_{target}.

Furthermore, this brings up a typical NP-hard problem. Each time when any $A_{eligible}$ is relocated to S_{target}, it is necessary for S_{target} to update the eligibility of all the activities and reselect the eligible activities. After an initial optimal $A_{eligible}$ is relocated to S_{target}, the subsequent choice of $A_{eligible}$ cannot guarantee the optimal accumulative time performance compared with other choices. Under the constraints of our approach, the optimal waiting time cannot be obtained unless all the possible choices of the $A_{eligible}$ combination are tried out and each of their performance relative to waiting time optimization is compared. Even given a solution of $A_{eligible}$ combination, the only way to find out whether it is optimal is still to enumerate all the possible $A_{eligible}$ combination and compare all the results in our approach at this point. This can be generalized that the optimality of a given solution of this problem cannot be ensured by any method to our knowledge within polynomial time. Therefore, to optimize time performance for a single subprocess is an NP-hard problem under our constraints of this approach.

To approach an optimal solution in this case, techniques of approximation need to be adopted in our algorithms. Initially, the absolute value of the contribution to time optimization from each $A_{eligible}$, namely AB_{time} is calculated as the waiting time of S_{target}, namely WA_{target} subtracting the working time of each $A_{eligible}$, namely $WO_{eligible}$. $A_{eligible}$s are sorted in an ascending order of their AB_{time}.

After the first element in $\{A_{eligible}\}$ is extracted, $\{A_{eligible}\}$, along with $\{WA_{target}\}$ and $\{AB_{time}\}$ are updated according to the varying pattern of the data delivery of S_{target}. After sorting $\{AB_{time}\}$, the smallest one is compared with WA_{target}. If it is larger than WA_{target}, then WA_{target} is believed to be the optimal waiting time of S_{target} and no further update is needed. Otherwise the above process repeats until a smaller WA_{target} is found. This algorithm, instead of enumerating all the possibilities, affirmatively select the observed optimal solutions to approach an approximately optimal waiting time.

The pseudo code of this algorithm is listed in Algorithm 1. In our approach, the details of sorting, *AST*, printing path, tailoring, and merging are not addressed because they are well-known sophisticated techniques.

Algorithm 1 Optimize_Time (Cluster_Subprocess, Waiting_Time): deriving the optimized time for a cluster subprocess

Input:
S_{target} := a cluster process that needs to optimize time performance,
$A_{eligible}$:= activity that is eligible for relocation to the cluster process that needs to optimize time performance,
$WO_{cluster}$:= working time of the cluster, a constant
WA_{target} := waiting time of the cluster process that needs to optimize time performance,
WO_{target} := working time of the cluster process that needs to optimize time performance,
$WO_{eligible}$:= working time of the eligible activities,
AB_{time} := absolute value of working time of the eligible activities subtracting waiting time of the cluster process that needs to be timely optimized,
Output:
Optimize_Time(S_{target}, WA_{target}) := optimized waiting time of S_{target}

```
1: $WA_{target}$ := absolute value of ($WO_{cluster}$ - $WO_{target}$);
2: $\{A_{eligible}\}$ := $\emptyset$;
3: $\{WO_{eligible}\}$ := $\emptyset$;
4: $\{AB_{time}\}$ := $\emptyset$;
5: for each $A \notin S_{target}$ in business process do
6:     check whether $A$ is forwardly moveable, backwardly moveable, or loosely-parallelled moveable to $S_{target}$;
7:     if yes then
8:         $\{A_{eligible}\}$ := $\{A_{eligible}\}$ + $A$;
9:         $AB_{time}$ := absolute value of ($WA_{target}$ - $WO_{eligible}$);
10:        $\{AB_{time}\}$ := $\{AB_{time}\}$ + $AB_{time}$;
11:        $\{WO_{eligible}\}$ := $\{WO_{eligible}\}$ + $WO_{eligible}$;
12:    end if
13: end for
14: sort($\{AB_{time}\}$);
15: sort($\{WO_{eligible}\}$) according to sort($\{AB_{time}\}$);
16: sort($\{A_{eligible}\}$) according to sort($\{AB_{time}\}$);
17: if $\{AB_{time}\}$[0] $<$ $WA_{target}$ and $\{A_{eligible}\} \neq \emptyset$ then
18:     $WO_{target}$ := $WO_{target}$ + $\{WO_{eligible}\}$[0];
19:     $S_{target}$ := $S_{target}$ + $\{A_{eligible}\}$[0];
20:     Optimize_Time($S_{target}$, $WA_{target}$);
21: else
22:     break;
23: end if
```

2) Time Optimization for The Overall Business Process: The technique of optimizing time performance of the overall business process is generally based on that of a single cluster subprocess. Instead of focusing on simply optimizing the working time in one single cluster subprocess, the techniques of time optimization for overall business process aims at reducing the total working time of the execution of the overall business process.

One major difference is the fact that AB_{time}, that is calculated simply as the absolute value of WA_{target} subtracting $WO_{eligible}$, cannot be used as the criteria to accurately reflect the overall time optimization. In fact, not only the contribution from $WO_{eligible}$ after $A_{eligible}$ being relocated to S_{target}, but also the impact that the relocation of $A_{eligible}$ imposes on its original subprocess needs to be considered to contribute to the varying time performance of the overall business process. Define $VA_{original}$ as the varying working time of the cluster where $A_{eligible}$ is originally located. $VA_{original}$ is calculated as the working time of the original cluster before $A_{eligible}$ being relocated, subtracting the working time of the original cluster after $A_{eligible}$ being relocated. Similarly, VA_{target} denotes the varying working time of the cluster that needs to optimize time performance and is calculated as the working time of that cluster after $A_{eligible}$ being relocated subtracting the working time of that cluster before $A_{eligible}$ being relocated.

VA_{sum}, defined as the summation of $VA_{original}$ and VA_{target}, is used to indicate how the relocation of $A_{eligible}$ changes the total time of the execution of the overall business process. If VA_{sum} is negative, then the total execution (working) time of the overall business process is decreased, and vice versa. For any $A_{eligible}$, the more negative VA_{sum} is, the more time optimization the overall business process gains by relocating the $A_{eligible}$.

Ideally, to achieve the time optimization of overall business process, it is equivalent to realizing the time optimization of each subprocess and sum them up. However, each time after one $A_{eligible}$ is relocated to a cluster subprocess, the subprocesses whose pattern of data delivery is impacted need to update their respective $\{A_{eligible}\}$. Moreover, the possibility of a relocated $A_{eligible}$ being selected and relocated by another subprocess at a later stage increases the uncertainty of identifying an optimal solution of reconstructing the business process. Therefore, to optimize time performance of the overall business process is at least as hard as that of a single subprocess and can be perceived as an NP-hard problem as well. To efficiently approach the approximated optimal time performance of the overall business process, an array, namely $\{A_{abandon}\}$ is defined to store the $A_{eligible}$ that has been relocated to make sure that even $A_{eligible}$ might be optimal for other subprocesses at a later stage, they would not be relocated anymore. This process ensures the reduction of the number of the remaining $A_{eligible}$s.

Our algorithm is initiated by detecting $\{A_{eligible}\}$ for each S_{target}. $\{A_{eligible}\}$ is then sorted along with $\{VA_{sum}\}$ in an ascending order. All the first elements of all the $\{VA_{sum}\}$ are then compared. The $A_{eligible}$ associated with the smallest VA_{sum} is relocated to its corresponding S_{target}. Then this $A_{eligible}$ is inserted to $\{A_{abandon}\}$, that indicates this $A_{eligible}$ cannot be relocated by any subprocess at a later stage. This process is repeated in our algorithm.

One S_{target} quits relocating $A_{eligible}$ when either its VA_{sum} associated with the $A_{eligible}$ is a positive value, that means the overall time of executing business process would be increased after the relocation of the $A_{eligible}$, or no $A_{eligible}$ can be found. The execution of our algorithm is terminated after each S_{target} quits its respective process.

The pseudo code of this algorithm is listed in Algorithm 2. The process of calculating $VA_{original}$ and VA_{target} is omitted here for simplicity.

V. Conclusion and Future Work

There are three factors in business process reengineering: improvement of quality, reduction of cost, and reducing time. In this paper we study a data-centric approach to optimize time performance in workflow-based business process. Assuming deterministic business activities whose components stay consistent, we focus on reconstructing the original business process by relocating the business activities in the business process. Our approach is initiated by representing a workflow-based business process with an abstract syntax tree based (*AST-based*) approach. By applying its properties, the eligibility for a business activity that can be relocated to a given subprocess is determined by whether it can be forwardly moveable, backwardly moveable, or loosely-parallelled moveable to a subprocess.

Furthermore, we explore how the time performance of a single subprocess can be optimized. It is observed that time inefficiency of one subprocess, that is reflected by waiting time in our approach, is generated when the execution of the subprocess is completed while others in the same cluster are still being executed. By collecting the eligible business activities, we develop algorithms to identify the ones that can approach the optimal (that is, minimal) waiting time performance of the associated subprocess and relocate them to that subprocess. Moreover, we develop algorithms to explore how to optimize time performance of the overall business process. Based on the methods to optimize time performance of a given subprocess, we approach the optimal time performance of the overall business process by modifying the criteria to determine the eligible activities that can contribute to reducing the working time of the overall business process. In addition, our design of restricting that any activity can only be relocated once efficiently derives the approximate optimal solution and avoids the potential NP-hard problems.

Algorithm 2 Optimize_Time (Business_Process, Waiting_Time): deriving the optimized time for the overall business process

Input:
BP := business process that needs to optimize time performance,
WO_{BP} := working time of the overall business process, apriori knowledge
S_{target} := a cluster process that needs to optimize time performance,
$A_{eligible}$:= activity that is eligible for relocation to the cluster process that needs to optimize time performance,
$A_{abandon}$:= activity that has already been relocated to the cluster process that needs to optimize time performance, $\{A_{abandon}\} := \emptyset$,
$VA_{original}$:= varying working time of the original cluster where the eligible activity is located, apriori knowledge for simplicity
VA_{target} := varying working time of the target cluster where the eligible activity is located, apriori knowledge for simplicity,
VA_{sum} := varying working time of the business process contributed by the eligible activity after being relocated,
$VA_{interim}$:= interim element to represent the the smallest VA_{sum} of all the subprocesses,
$A_{interim}$:= interim element to represent the $A_{eligible}$ corresponding to its corresponding $VA_{interim}$,
Output:
Optimize_Time(*BP*, WO_{BP}) := the optimized working time of the overall business process

```
1: for each $S_{target} \in$ BP do
2:     $\{A_{eligible}\} := \emptyset$;
3:     $\{VA_{sum}\} := \emptyset$;
4:     $\{VA_{interim}\} := \emptyset$;
5:     $\{A_{interim}\} := \emptyset$;
6:     for each $A \notin (S_{target} \cup \{A_{abandon}\})$ in BP do
7:         check whether A is forwardly moveable, backwardly moveable, or
           loosely-parallelled moveable to $S_{target}$;
8:         if yes then
9:             $\{A_{eligible}\} := \{A_{eligible}\} + A$;
10:            $VA_{sum} := VA_{target} + VA_{original}$;
11:            $\{VA_{sum}\} := \{VA_{sum}\} + VA_{sum}$;
12:        end if
13:    end for
14:    sort($\{VA_{sum}\}$);
15:    sort($\{A_{eligible}\}$) according to sort($\{VA_{sum}\}$);
16:    $\{VA_{interim}\} := \{VA_{interim}\} + \{VA_{sum}\}[0]$;
17:    $\{A_{interim}\} := \{A_{interim}\} + \{A_{eligible}\}[0]$;
18: end for
19: sort($\{VA_{interim}\}$);
20: sort($\{A_{interim}\}$) according to sort($\{VA_{interim}\}$);
21: if $\{VA_{interim}\}[0] < 0$ and $\{A_{interim}\} \neq \emptyset$ then
22:     $WO_{BP} := WO_{BP} + \{VA_{sum}\}[0]$;
23:     BP updates $\{A_{interim}\}[0]$;
24:     $\{A_{abandon}\} := \{A_{abandon}\} + \{A_{interim}\}[0]$;
25:     Optimize_Time(BP, $WO_{BP}$);
26: else
27:     break;
28: end if
```

There are generally two domains that can be adopted to evaluate our approach. One is to develop realistic software systems to implement our approach for optimizing time performance. Moreover, the performance of our approximated approaches for the time optimization of both a single subprocess and the overall business process can be verified and potentially improved with the support of the realistic system implementation. As a matter of fact, many enterprise software systems, such as ERP and BPM, have been widely used and proved to be the pioneers for applying the philosophies of business process optimization. Their users, including but not limited to, the enterprise managers and the enterprise software designers and developers can be the sources in providing valuable feedback for improving the efficacy of our approach in the future. Accordingly, we plan to conduct case studies for them to refine our perspectives and approaches by collecting their valuable suggestions.

Time optimization could be complicated in terms of different scenario settings, in which the corresponding approaches can be significantly different from each other. Assuming consistent business activities, our approach at this point focuses on the inter-activity adjustment in a business process by relocating the eligible activities to subprocesses. In the future, methods for the intra-activity adjustment can be proposed and implemented to realize time optimization in the case when redesigning business activities tend to be necessary. More domain-specific methods can be proposed for different types of business processes that are relevant to specific industries, such as software engineering, chemical engineering, etc. as well.

Business process optimization approaches can provide the direct benefit for business goals to optimize the specified objectives. Researches with respect to business process optimization in reality can be complicated and some relevant elements are hard to be theoretically modeled. Therefore, it is seemingly difficult to come up with unified solutions for multi-aspect problems. How to improve the value of the research of business process optimization remains a long-term challenge and interest. Starting from our approach to optimize time performance, in the long run, we would pay attention to discovering more properties under different settings of business processes and aim at proposing corresponding solutions to realize their optimization.

References

[1] 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.

[2] Wil M. P. Van Der Aalst, Arthur H. M. Ter Hofstede, and Mathias Weske. Business process management: A survey. In *Proceedings of the 2003 International Conference on Business Process Management*, BPM'03, pages 1–12, Berlin, Heidelberg, 2003. Springer-Verlag.

[3] Yonghua Zhou and Yuliu Chen. Project-oriented business process performance optimization. In *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, volume 5, pages 4079–4084 vol.5, Oct 2003.

[4] Hajo A. Reijers. Product-based design of business processes applied within the financial services. *Journal of Research and Practice in Information Technology*, 34(2):110–122, 2002.

[5] Ingo Hofacker and Rudolf Vetschera. Algorithmical approaches to business process design. *Computers and OR*, 28(13):1253–1275, 2001.

[6] Christodoulos Floudas and Xiaoxia Lin. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Annals of Operations Research*, 139(1-4):131 – 162, 2005.

[7] Heinrich Rommelfanger. The advantages of fuzzy optimization models in practical use. *Fuzzy Optimization and Decision Making*, 3(4):295–309, December 2004.

[8] Jose M. Pinto and Ignacio E. Grossmann. Assignment and sequencing models for the scheduling of process systems. *Annals of Operations Research*, 81(1-4):433 – 466, 1998.

[9] Carlos A. Mendez, Jaime Cerd, Ignacio E. Grossmann, Iiro Harjunkoski, and Marco Fahl. State-of-the-art review of optimization methods for short-term scheduling of batch processes. *Computers and Chemical Engineering*, 30(6-7):913–946, 2006.

[10] Andreas T. Ernst, Houyuan Jiang, Mohan Krishnamoorthy, and David Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27, 2004.

[11] R.S.Ruth Sara Aguilar-Saven. Business process modelling: Review and framework. *International Journal of Production Economics*, 90(2):129–149, july 2004.

[12] 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.

[13] 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.

[14] 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.

[15] Yuqun Zhang and Dewayne E. Perry. A goal-directed method towards business process modeling. In *Service-Oriented System Engineering, 2014. IEEE International Conference on (To Appear)*, volume 12, Apr 2014.

[16] 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.

[17] Dewayne E. Perry, Nancy A. Staudenmayer, and Lawrence G. Votta. Understanding software development: Processes, organisations and technologies. *IEEE software*, 11:36–45, 1994.

[18] 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.

[19] Object Management Group (OMG). Business process model and notation (bpmn) version 2.0. Technical report, jan 2011.