Optimization of BPMN Processes via Automated Refactoring

Francisco Durán[1] and Gwen Salaün[2]

[1] ITIS Software, University of Málaga, Málaga, Spain
[2] Univ. Grenoble Alpes, CNRS, Grenoble INP, Inria, LIG, F-38000 Grenoble France

Abstract. Business process optimization has become a strategic aspect of companies' management due to the potential of cost reduction and throughput improvement. There are several ways to achieve process optimization, depending on the level of expressiveness of the processes at hand. In this paper, we focus on processes described using BPMN, but also including an explicit description of execution time and resources associated with tasks. We propose a refactoring procedure whose final goal is to reduce the total execution time of the process given as input. Such a procedure relies on refactoring operations that reorganize the tasks in the process by taking into account the resources used by those tasks. This process refactoring technique is fully automated by a tool that we implemented and applied on several examples for validation purposes.

1 Introduction

Context. Business process optimization is a strategic activity in organizations because of its potential to increase profit margins and reduce operational costs. Optimization is however a difficult task to be achieved manually since several parameters should be taken into account (execution times, resources, costs, etc.). These parameters are not systematically included in existing languages used for modelling and managing business processes. Moreover, optimization requires a high level of expertise that not all users have. Automated techniques are thus required to optimize a given process for certain criteria of interest.

In this work, we assume that a description of a business process is given using the standardized workflow modelling language BPMN. This language allows us to define the set of tasks involved in a process and the order in which they should be executed. This behavioural description of the model can be extended with information on the time each task takes to execute and an explicit description of the resources required for executing each task. As a consequence, the resulting model of the process does not only take into account behavioural aspects but also quantitative aspects.

Motivations. Processes are not built once and for all in a monolithic way. During their life time, processes have to be changed or updated for several possible reasons: addition or suppression of some specific task, improvement of the process with respect to a given criterion (e.g., overall execution time), adjustment of the process to consider a new regulation or internal directive, etc. When writing a process or when updating it as suggested before, the quality or correctness of the process has to be

preserved. This might not be the case if process writing or reengineering/refactoring is achieved manually by human beings. Moreover, refactoring steps may be very difficult to apply when BPMN models also take into account quantitative aspects and when some criteria, such as resource usage or execution time, are used to guide the refactoring steps so as to generate an optimal process (e.g., the process must execute as quickly as possible). Therefore, there is a need for automated refactoring techniques in order to generate an optimal version of a process during its writing or update.

Proposal. Given a process model, we propose some optimization techniques that rely on the refactoring of the given process. By changing the structure, we aim at generating a different process whose overall execution time is reduced compared to the original process. The main idea is to increase the level of parallelism of the tasks involved in the process. The refactoring steps should however be applied with care. For instance, it is not helpful to put in parallel two tasks using a same resource since they will compete for such a resource and a resource cannot be involved in two different tasks at the same time. Moreover, it does not always make sense to put two tasks in parallel, and in some cases such causal dependencies must be preserved (e.g., some product must be packaged before its delivery).

More precisely, we propose in this paper optimization techniques based on process refactoring. Our approach takes as input a BPMN process extended with time and resources associated with tasks, and generates as output a new version of this process. To do so, we first analyze the process to identify tasks that could be executed earlier because the resources that one of these tasks requires for its execution are available before its execution begins. Alternative processes are generated by moving these tasks backwards (closer to the initial node). Our approach works using an iterative approach. Each newly generated process is similarly analyzed, and new alternative processes are generated in the same way. Since many processes can be synthesized by our approach, there are different ways to generate and handle these new processes. We have implemented and carried out experiments with two strategies: (i) an exhaustive exploration of all possible processes generated by refactoring, and (ii) a guided exploration of the new processes by using some heuristic. All these techniques are fully automated in a tool that we implemented and which has been validated on many examples.

Organization. The rest of this paper is organized as follows. Section 2 introduces the considered subset of the BPMN notation and its extension with time and resources. Section 3 provides an overview of the different steps of our approach. Section 4 focuses on the refactoring process to change the structure of the process. Section 5 presents the tool support and some experimental results to assess the accuracy and performance of our approach. Section 6 compares our solution to related work and Section 7 concludes.

2 BPMN with Time and Resources

BPMN 2.0 (BPMN, as a shorthand, in the rest of this paper) was published as an ISO/IEC standard [10] in 2013 and is nowadays extensively used for modeling and developing business processes. In this paper, we focus on activity diagrams including the BPMN constructs related to control-flow modeling and behavioural aspects.

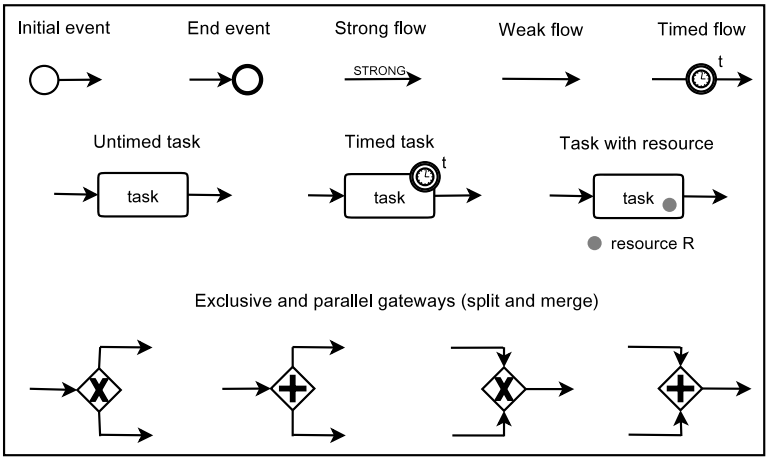

Fig. 1. Supported BPMN syntax

Beyond those constructs, execution time and resources are also associated with tasks. Figure 1 summarizes the BPMN constructs supported in this work.

Specifically, the node types *event*, *task*, and *gateway*, and the edge type *sequence flow* are considered. Start and end events are used, respectively, to initialize and terminate processes. A task represents an atomic activity that has exactly one incoming and one outgoing flow. A sequence flow describes two nodes executed one after the other in a specific execution order. A task and a flow may have a duration or delay. The timing information associated to tasks and flows is described as a literal value (a non-negative real number, possibly 0). Resources are explicitly defined at the task level. A task that requires resources can include, as part of its specification, the required resources. Information about time and resources can be used jointly for a given task. In such a case, it means that the task needs those resources to be able to execute, and once the resources are acquired, the task is going to execute for the specified duration.

Sequence flows can be of two possible types, to explicitly specify flows that must be preserved during the refactoring process. A *strong* flow corresponds to a causal dependency between two nodes that cannot be changed (e.g., some product must be packaged before its delivery). A *weak* flow corresponds to a loose connection between two nodes that may be preserved or not (e.g., the product could be delivered before the client pays for it).

Gateways are used to control the divergence and convergence of the execution flow. We consider in this work the two main kinds of gateways used in activity diagrams, namely, *exclusive* and *parallel* gateways. Gateways with one incoming branch and multiple outgoing branches are called *splits*, e.g., split parallel gateway. Gateways with one outgoing branch and multiple incoming branches are called *merges*, e.g., merge parallel gateway. An exclusive gateway chooses one out of a set of mutually exclusive alternative incoming or outgoing branches. A parallel gateway creates concurrent flows for all its outgoing branches or synchronizes concurrent flows for all its incoming branches.

3 Overview of the Approach

In this section, we give an overview of the different steps of our approach. We start by introducing the simulation-based analysis with which possible improvements in the structure of the process in terms of resource usage are identified. Then, we present the refactoring procedure and how different strategies are used to explore the possible solutions.

3.1 Simulation-Based Analysis

In this work, the main idea of the refactoring approach is to change the structure of the process in order to reduce its total execution time. Since optimization mostly targets process execution time, we need to compute this time for a given process. To do so, we rely on simulation-based techniques which turn out to simplify the computation of execution times in the presence of resources. Indeed, a task needs to acquire the required resources to be able to execute, and if the resources are not available the task cannot execute. The competition for resources may thus induce delays, and these delays are not easy to identify. To analize the process, we simulate it a certain number of times (this is a parameter of the approach). During these executions, some information about the execution of tasks (pending, executing, completed) and the usage of resources is stored. At the end of each execution, we store the time taken for completing the process. The average of those times allows us to compute the average execution time of the process.

After completion of the simulation, there is an analysis step which explores the simulation log for retrieving specific information. In particular, we look for specific timestamps during the simulation at which a task is in a pending state (meaning that this task is still waiting to be able to execute), and all resources required for executing this task are available. This means that this specific task could execute earlier in a process, and we will use this information to change the structure of the process by trying to move this task backwards in the process. This analysis step returns as output a set of tasks that could be executed earlier. Each task in this set verifies the aforementioned constraints (pending task but with required resources available) during a period of time of the simulation.

3.2 Refactoring Procedure

A refactoring step takes as input a BPMN process and a task that can be executed earlier in that process. Then, depending on the type of the preceding node, a specific refactoring operation is applied. Such refactorings represent changes in fragments of the process, and multiple refactoring operations may be applied consecutively. To do so, the refactoring procedure relies on the generation and exploration of possible solutions and keeps track of the visited processes to avoid recomputations.

The refactoring operations are presented in Section 4. A refactoring operation can be seen as a model transformation $L \Rightarrow R$ *if* C, with L and R subprocess patterns and C a condition on them. Each refactoring operation results in the transformation of a process (or model) into another. Thus, a refactoring operation can be applied if its left-hand side L matches a subprocess (or submodel) of the current process P, that is, $P|_p = L\sigma$, for some match σ and position p, and the condition C is

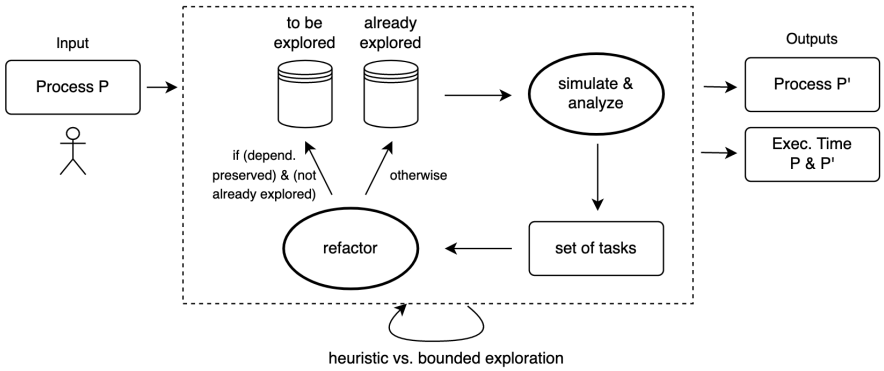

Fig. 2. Overview of the approach

satisfied for that match, that is, if $C\sigma$ is evaluated to true. The application of such a refactoring rule results in the replacement of the matched subprocess $P|_p$ by the right-hand side $R\sigma$. By repeatedly applying the available refactoring transformations on different parts of a process, new processes are generated. Strategies are thus necessary to explore all these new processes, and eventually return the optimal one.

Figure 2 depicts the input and outputs as well as the main steps of the refactoring process from a global perspective. The approach takes as input a BPMN process that includes a description of time and resources as shown in Section 2. This process is moved to the queue of processes to be explored. One process is then extracted from this queue (the original one in the first iteration) and the simulation-based analysis is carried out to identify the tasks that could be executed earlier. For each of these tasks, the corresponding refactoring operation is applied, generating a new process. Since the refactoring operation just moves the given task one step backwards, the resulting process may require additional changes. Therefore, if the resulting process has not been explored yet, and if the resulting process respects the strong dependencies defined in the original process (causal dependencies corresponding to strong flows must be maintained by the refactoring process), this new process is moved to the queue of processes to be explored. Otherwise, it is moved to the queue of processes already explored.

The refactoring procedure consists in the iterative application of refactoring operations. Since there are different ways to refactor a process, it is not possible to know at the beginning of the approach which refactoring step will lead to the best result (process with the minimal execution time). The exhaustive exploration of the search space, achieved by repeatedly attempting each possible refactoring operation on the tasks that may be moved backwards, may be time-consuming. To reduce the exploration space, we propose two different strategies:

- The first strategy consists in a *bounded breadth-first search*. It carries on an exhaustive exploration of all processes to be explored up to a certain bound given as parameter. For each possible task to be moved earlier in the process, the corresponding refactoring operation is applied. If the resulting process has not been

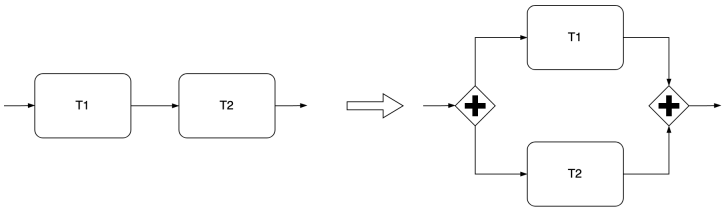

Fig. 3. Sequence of tasks

analyzed before, it is placed in the to-be-explored queue. The procedure continues while there are processes in such a queue and the bound has not been reached.

– The second strategy, instead of exploring the search graph by applying refactoring operations on all tasks that can be moved earlier, only expands by applying refactoring on the task closer to the start event in the BPMN process. The intuition is that by moving this task backwards, it will be placed closer to the initial event, thus reducing the number of times it may be moved in the future. In other words, we try to move first the tasks closer to their final positions in the process. This strategy is referred as *heuristic-based* in the rest of this paper.

Several experiments showing the behaviour of the following exploration algorithms, including the use of these two strategies are presented in Section 5.

4 Refactoring Operations

This section presents a set of refactoring operations. In each of these refactoring operations, given as input a process and a task that has to be moved earlier in the process, a new process is returned as output. The refactoring operation to be applied depends on what type of node precedes the task to be moved backwards. There are actually three main cases: this node can be another task, a merge node, or a split node. Therefore, we will organize the rest of this section tackling successively these three cases. For each case, the proposed refactoring operations are presented. Before starting, we also recall that this refactoring step focuses on the process structure and on the usage of resources by tasks, but does not take into account strong flows, which is handled at another level (as explained in Section 3.2).

4.1 Task

This first case is rather straightforward. If a task can be executed earlier, and that task is preceded by another task, we check whether these two tasks are sharing some resources. If they do not share any resources, the process is transformed to execute these two tasks within a common parallel gateway, as illustrated in Figure 3.

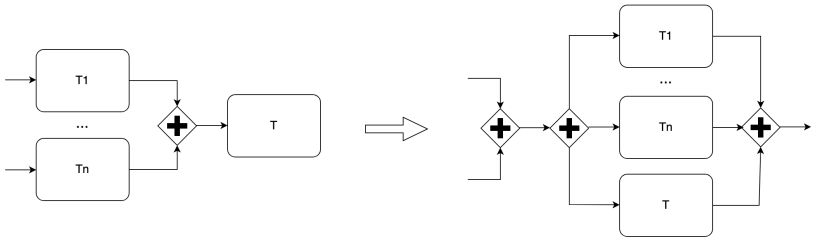

Fig. 4. Merge parallel gateway with preceding tasks (no shared resources)

4.2 Merge Node

If the task `T` to be moved earlier is preceded by a merge node, there are several possible (sub)cases. First of all, this node can be an exclusive or a parallel gateway. Second, before that merge, there may be only tasks, only other merge nodes, or a combination of both. We have specific operations for each of these cases. We will introduce these operations in the rest of this subsection.

Let us first consider the case in which the merge parallel gateway is preceded by a set of tasks `T1`...`Tn`, and none of these tasks share resources with the task `T` (left-hand side of Figure 4). In that case, all tasks are gathered in parallel just before the merge parallel gateway (right-hand side of Figure 4). Note that a split parallel gateway is added before these tasks in order to avoid that `T` executes before the tasks preceding the tasks before the merge. The preceding tasks might be using the same resources.

Assume now that the tasks preceding the merge parallel gateway share some resources with task `T`. If they all share resources, no refactoring is possible (since adding an additional task in parallel, competing for the same resources, would not improve the process execution time). If only one task shares resources with `T`, then `T` is moved after that task but before the merge as shown in Figure 5. If there are several tasks sharing resources with `T` (but not all), then `T` is moved before the merge and right after an additional merge parallel gateway for this subset of resources. Figure 6 shows such a case where `T1`...`Ti` share some resources with `T`. Therefore, after the refactoring operation, `T` appears after `T1`...`Ti` whereas `Ti+1`...`Tn` keep executing in parallel.

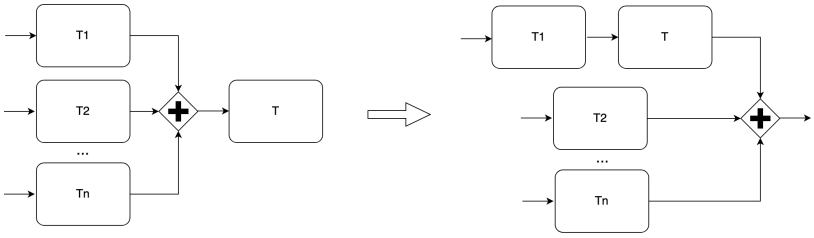

Fig. 5. Merge parallel gateway with preceding tasks (shared resources with one task)

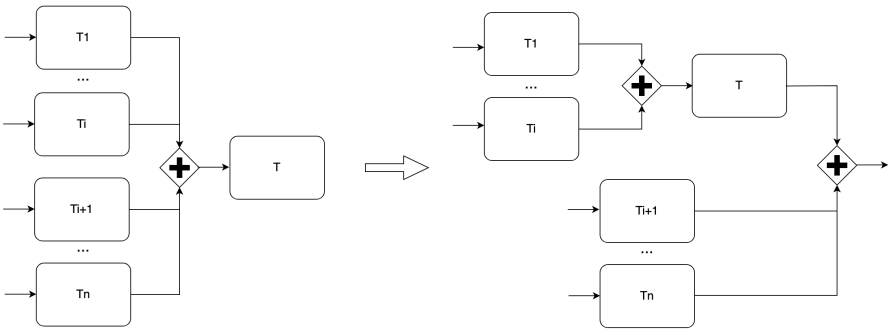

Fig. 6. Merge parallel gateway with preceding tasks (shared resources with several tasks)

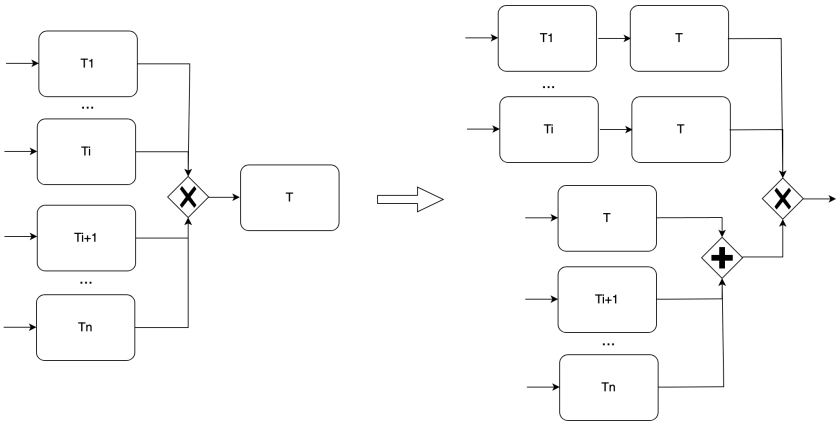

Fig. 7. Merge exclusive gateway with preceding tasks

If what precedes the task to be moved is an exclusive gateway, and if there are only tasks before the merge, there are two cases. For each task before the merge, if task `T` shares some resource with that task, `T` is moved before the merge but after that task. If task `T` does not share any resource with that task, `T` is put in parallel with that task. Figure 7 illustrates this operation, showing that in the case of `T1...Ti` (shared resources), `T` is moved after each of them, whereas in the case of `Ti+1...Tn` (no shared resources), they all appear in parallel in the resulting process. Note that `T` appears multiple times in the resulting process, because by including `T` in an exclusive pattern, it has to be executed once by each existing branch to maintain the intended behaviour.

As far as cascading merges are concerned, we support cascading merge exclusive gateways (possibly finishing with a merge parallel gateway), by applying several times the patterns introduced above. However, if there are cascading merge parallel gateways or a merge parallel gateway followed by a merge exclusive gateway, refactoring is too complicated and is not applied. Let us take the example of two merge parallel gateways. If we move a task within the first one, this is ok as presented

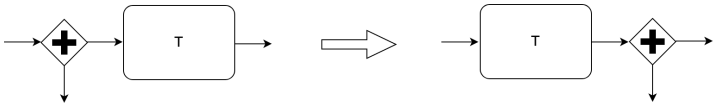

Fig. 8. Split parallel gateway

earlier. However, if there are other merge parallel gateways inside the first one, we do not know where to move that task since we do not want to execute it multiple times.

4.3 Split Node

If the task `T` to be moved earlier is preceded by a split parallel gateway, then `T` is moved before the split, whatever precedes the split (task, split or merge). Figure 8 illustrates this pattern by moving the task before the split parallel gateway.

If the task `T` is preceded by a split exclusive gateway, we apply refactoring only if the split is preceded with a task (`T1` for instance). In that case, we need to analyze the process to look for a merge gateway corresponding to the aforementioned split gateway. If the subprocess is balanced and the corresponding merge gateway is found, we still need to look at the resources used in that part of the process. If all tasks between the split and the merge do not share any resources with `T1`, then the whole block is moved in parallel with `T1`, as illustrated in Figure 9, because we cannot dissociate the contents (tasks for instance) appearing in the same branch of an exclusive structure. If the task `T` is preceded by a split exclusive gateway, and the preceding node is not a task (it is another split for instance), there is no simple refactoring and we keep it as is. Note that if the exclusive split preceding `T` is not balanced or some task in that block uses any of the resources of `T`, then the refactoring operation is not applied because optimization is not possible.

5 Implementation and Experiments

This section presents the tool support, a case study, and some experiments. Additional details about the tool and dataset used for the experiments are available online [1].

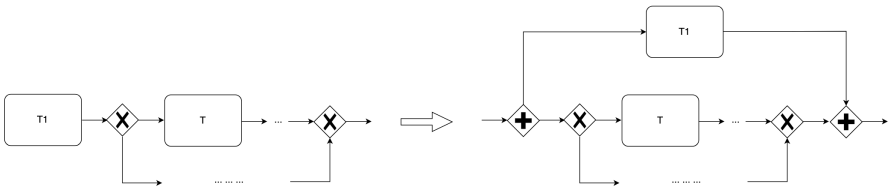

Fig. 9. Split exclusive gateway

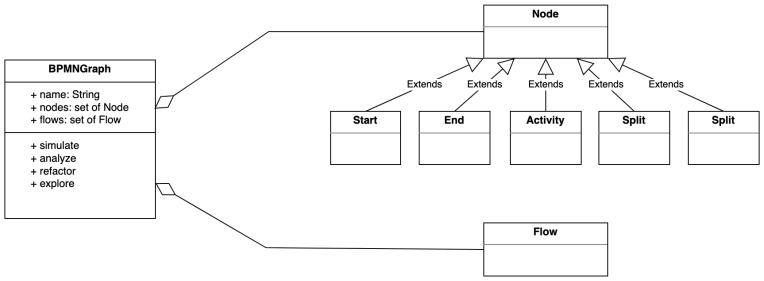

Fig. 10. Simplified class diagram

5.1 Tool

A tool implementing the above refactoring operations and their systematic application following the described procedures has been developed in Python. It works as presented in Figure 2: the user provides as input a BPMN process and obtains as output an optimal version of that process with the gain in terms of process execution time. The implementation consists of several classes as illustrated in Figure 10. The core of the approach is implemented in several methods dedicated to process simulation and analysis (of the simulation), refactoring operations, and exploration of the refactored processes by using different strategies. Processes to be explored and already explored are stored into dictionaries. Hash values for processes are used as keys and are computed using the number of nodes at distance 1, 2, 3, etc. from the start event. These numbers are then concatenated to form a key. Since we may have collisions, for each hash value, we store a list of processes in the dictionary. As for the transformation from BPMN XML to our encoding of BPMN into Python (in both directions), we take advantage of the transformation capabilities available in VBPMN [12, 13].

5.2 Case Study

We illustrate our approach with a process describing the business trip organization given in Figure 11. Each task has an annotation with a pair of values where the first value is the duration of the task (in days for instance) and the second value is the set of resources required to execute that task. Let us introduce the process with more details. First, the assistant fills in the required documents. Then, the travel agency is in charge of booking flight tickets, followed by the reservation of accommodation by the user in that case. Visa is then prepared and, in parallel, the user has to be vaccinated. The final part of the process is executed when the user is back from the trip. All necessary documents for reimbursement are returned by the user. Reimbursement is then completed by the financial staff. Finally, all documents are archived by the assistant. It is worth noting that there are several strong flows in the original process, before and after `g2`, and between task `ReturnDocuments` and task `Reimbursement`. This means that the causal dependency between these tasks is important and must be preserved by the refactoring process.

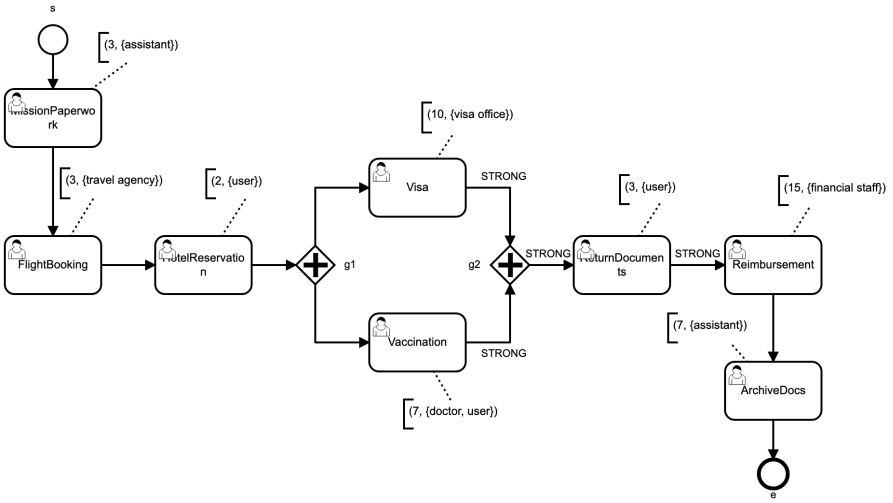

Fig. 11. Example of "Trip Organization" process in BPMN

To compute the refactored version of this process, we use a bounded exploration (with 300 as bound). The resulting process is given in Figure 12, and was obtained after about 130 iterations. It takes about 20 seconds to compute the resulting process. The execution time of all the tasks of the original process is 43 days whereas the new version executes in 28 days.

Let us now comment on this new version of the process. We can see that causal dependencies defined by strong flows are preserved in this process: tasks `Visa` and `Vaccination` are executed before task `ReturnDocuments`, and task `ReturnDocuments` is executed before task `Reimbursement`. In the first part of the process, we can see that several tasks can be executed in parallel because they all use different resources. However, tasks `HotelReservation` and `Vaccination` cannot be executed in parallel because they use the same resource `user`. After task `ReturnDocuments`, we can see that the two final tasks are executed in parallel because they use different resources.

5.3 Experiments

In this section, we show some experimental results obtained when applying our tool to different BPMN processes. The main goal of this section is to evaluate our tool in terms of performance and accuracy of the results, particularly comparing both strategies (bounded and heuristic-based exploration).

Table 1 shows some experiments on ten processes, mostly taken from the literature, e.g., BPMN processes introduced in [4–6], and from the VBPMN database of examples [12, 13]. The table first characterizes each process in terms of number of tasks, flows, gateways, and strong flows (SF). Then, the table gives the current Average Execution Time of the process (AET_c) and the Average Execution Time corresponding to the optimal process (AET_b). The optimal process corresponds

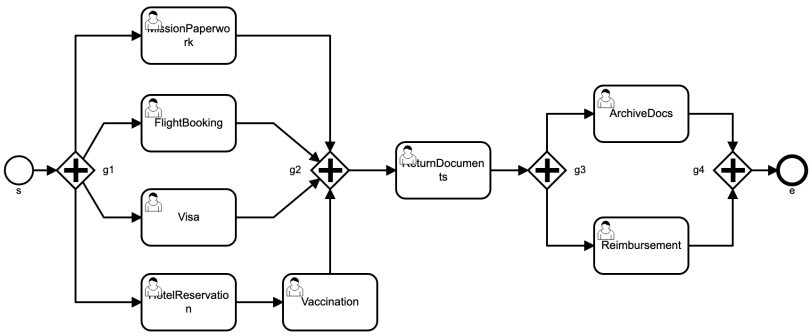

Fig. 12. Trip Organization process after automated refactoring

to the process with the shortest execution time. This optimal process and the corresponding execution time were built and computed manually by the authors of this paper. Then, we show the results for the bounded exploration (bound fixed to 300) and for the heuristic-based approach. For each option, we give the execution time of the final process and the computation time to obtain the result.

Let us now comment on the results shown in this table. First of all, we can see that the bounded exploration succeeds in most cases to find the optimal solution, that is, the process with the shortest execution time. For larger examples (e.g., processes 7, 9 and 10 in the table), the best solution is not found because the bound (300) was too small to explore enough processes and find the best solution. However, if we increase the bound, the best solution is eventually found. As an example, for row 7, the best solution (with 18 as execution time) is found after about 750 iterations.

Table 1. Experimental results

BPMN	Characteristics						Bounded Explo.		Heuristic	
Proc.	Tasks	Flows	Gateways	SF	AET_c	AET_b	AET	Time	AET	Time
1	5	13	4✚	0	70	50	50	1s	50	1s
2	5	15	6✖	0	40	25	25	2s	25	2s
3	6	13	2✚ 2✖	0	19	6	6	2s	9	1s
4	8	12	2✚	4	43	28	28	21s	35	1s
5	8	12	2✚	2	21	14	14	23s	16	1s
6	9	10	0	6	90	50	50	31s	90	1s
7	10	20	6✚	0	24	18	19	38s	20	1s
8	10	20	2✚ 4✖	3	23	13	13	67s	17	1s
9	12	29	8✚	0	200	120	140	50s	120	1s
10	15	36	10✚	0	260	180	220	103s	260	3s

As for the heuristic-based strategy, there are several cases for which the best solution is not found, but in very specific cases (e.g., process 9), it may give better results. Regarding computation time, the heuristic-based strategy is much faster (a few seconds) whereas the bounded exploration takes more time because it explores possibly many solutions (300 at most here). Regarding the computation time for the bounded exploration, we can see that this time increases with the size of the process (it takes more time to simulate, analyze and refactor a larger process than a simpler one). The number of refactorings does not really impact the computation time, since a bound is used to stop the exploration of potential solutions. Finally, the number of strong flows tends to reduce the number of possible refactorings thus the computation time, because strong flows can be seen as additional constraints on the process.

6 Related Work

This section starts with a short overview of extensions of BPMN with time and resource features, before presenting and comparing our approach with existing solutions for BPMN refactoring. Several works propose extensions of BPMN with time constructs, see, e.g., [2, 9]. In [9], the authors present Time-BPMN, an extension of BPMN that allows the specification of temporal constraints and dependencies within a BPMN diagram. In [2], a metamodel-based approach to integrate temporal constraints and dependencies is introduced. The time aspects are specified using rules and OCL constraints capture the semantics of these rules. Our solution shares similarities with the approach proposed in [9].

As far as resource allocation is concerned, several solutions have been proposed by the research community in the business process domain. Schömig and Rau [18] use colored stochastic Petri nets to specify and analyze business processes in the presence of dynamic routing, simultaneous resource allocation, forking/joining of process-control threads, and priority-based queueing. Li *et al.* [15] introduce *multidimensional workflow nets* to model and analyze resource availability and workload. Oliveira *et al.* [16] use generalized stochastic Petri nets for correctness verification and performance evaluation of business processes. In this work, we propose to associate resources to tasks, which is a flexible solution for modelling many different situations and scenarios. Moreover, our approach does not focus on the computation of metrics but changes the process structure to actually optimize some of these metrics.

Let us now focus on existing works on process refactoring. [19] presents six common mistakes made by developers when modelling with BPMN: inconsistent naming, large process diagrams, inconsistent use of gateways, inconsistent use of events, inconsistent use of loops, poor diagram layout. For each problem, the authors present best practices for avoiding these issues. As an example, the authors propose to use explicit gateways instead of using multiple incoming/outgoing sequence flows. [3] presents a technique for detecting refactoring opportunities in process model repositories. The technique works by first computing activity similarity and then computing three similarity scores for fragment pairs of process models. Using these similarity scores, four different kinds of refactoring opportunities can be systematically identified. As a result, the approach proposes to rename activities or to introduce

subprocesses. IBUPROFEN, a business process refactoring approach based on graphs, is presented in [8, 17]. IBUPROFEN defines a set of 10 refactoring algorithms grouped into three categories: maximization of relevant elements, fine-grained granularity reduction, and completeness. All these works mostly focus on syntactic issues and propose synctatic improvements of the process by, for instance, removing unreachable nodes or by merging consecutive gateways of the same type. They do not aim at providing any kind of optimization regarding the process being designed as we do.

In [14], the authors present an approach for optimizing the redesign of process models. It is based on capturing process improvement strategies as constraints in a structural-temporal model. Each improvement strategy is represented by a binary variable. An objective function that represents a net benefit function of cost and quality is then maximized to find the best combination of process improvements that can be made to maximize the objective. The BPMN subset used in [14] is very similar to the one we use in this paper. However, the approach is rather different since they compute optimal redesigns with respect to some constraints, whereas we propose refactoring patterns with respect to process execution times.

Last but not least, it is worth mentioning recent works providing support for building (optimal) processes. [7] proposes a semi-automated approach for helping non-experts in BPMN to model business processes using this notation. Alternatively, [11] presents an approach which combines notes taking in constrained natural language with process mining to automatically produce BPMN diagrams in real-time as interview participants describe them with stories. In this work, we tackle this issue from a different angle since we assume that an existing description of the process exists and that we want to automatically optimize it by updating its structure.

7 Concluding Remarks

In this paper, we have focused on a version of BPMN including task durations and an explicit description of resources. We have then proposed a simulation-based approach that allows us to identify some specific tasks which are waiting for being executed but for which the required resources are available. This means that, from the point of view of the process structure, these tasks could be executed earlier in the process. We then apply some refactoring transformations to move those tasks backwards in the process structure. This approach works by successively applying these refactorings and by thus exploring the possible solutions to find the optimal one. Several strategies have been implemented and vary in their way to apply these iterations. In any case, the refactoring process completes and returns as result a process whose average execution time is lower (or equal) than the one of the original process. Note that if the original process is already optimal, that process and its corresponding execution time are returned as output. All the steps of the refactoring approach are fully automated by a tool we implemented and applied on many examples of processes for validation purposes.

Acknowledgments. F. Durán has been partially supported by projects PGC2018-094905-B-100 and UMA18-FEDERJA-180, and by Universidad de Málaga, Campus de Excelencia Internacional Andalucía Tech. This work was also supported by the Région Auvergne-Rhône-Alpes within the "*Pack Ambition Recherche*" programme.

References

1. Workflow Refactoring Tool and Examples – (Blinded) GitHub Repository, 2022. https://github.com/afjdm/workflow-refactoring.
2. C. Arévalo, M. J. E. Cuaresma, I. M. Ramos, and M. Domínguez-Muñoz. A Metamodel to Integrate Business Processes Time Perspective in BPMN 2.0. *Information & Software Technology*, 77:17–33, 2016.
3. R. M. Dijkman, B. Gfeller, J. M. Küster, and H. Völzer. Identifying Refactoring Opportunities in Process Model Repositories. *Inf. Softw. Technol.*, 53(9):937–948, 2011.
4. F. Durán, C. Rocha, and G. Salaün. Stochastic Analysis of BPMN with Time in Rewriting Logic. *Sci. Comput. Program.*, 168:1–17, 2018.
5. F. Durán, C. Rocha, and G. Salaün. A Rewriting Logic Approach to Resource Allocation Analysis in Business Process Models. *Sci. Comput. Program.*, 183, 2019.
6. F. Durán and G. Salaün. Verifying timed BPMN processes using Maude. In *Proc. of COORDINATION*, volume 10319 of *LNCS*, pages 219–236. Springer, 2017.
7. Y. Falcone, G. Salaün, and A. Zuo. Semi-automated Modelling of Optimized BPMN Processes. In *Proc. of SCC'21*, pages 425–430. IEEE, 2021.
8. M. Fernández-Ropero, R. Pérez-Castillo, and M. Piattini. Graph-Based Business Process Model Refactoring. In *Proc. of the 3rd Int. Symposium on Data-driven Process Discovery and Analysis*, volume 1027 of *CEUR Workshop Proceedings*, pages 16–30, 2013.
9. D. Gagné and A. Trudel. Time-BPMN. In *Proc. of CEC'09*, pages 361–367. IEEE Computer Society, 2009.
10. ISO/IEC. International Standard 19510, Information technology – Business Process Model and Notation. 2013.
11. A. Ivanchikj, S. Serbout, and C. Pautasso. From Text to Visual BPMN Process Models: Design and Evaluation. In *Proc. of MoDELS'20*, pages 229–239. ACM, 2020.
12. A. Krishna, P. Poizat, and G. Salaün. VBPMN: Automated Verification of BPMN Processes. In *Proc. of IFM'17*, volume 10510 of *LNCS*, pages 323–331. Springer, 2017.
13. A. Krishna, P. Poizat, and G. Salaün. Checking Business Process Evolution. *Sci. Comput. Program.*, 170:1–26, 2019.
14. A. Kumar and P. Indradat. Optimizing Process Model Redesign. In *Proc. of ICSOC'16*, volume 9936 of *LNCS*, pages 39–54. Springer, 2016.
15. J. Li, Y. Fan, and M. Zhou. Performance Modeling and Analysis of Workflow. *IEEE Transactions on Systems, Man, and Cybernetics*, 34(2):229–242, Mar. 2004.
16. C. Oliveira, R. Lima, H. Reijers, and J. Ribeiro. Quantitative Analysis of Resource-Constrained Business Processes. *Trans. on Syst., Man, and Cybern.*, 42(3):669–684, 2012.
17. R. Pérez-Castillo, M. Fernández-Ropero, and M. Piattini. Business process model refactoring applying IBUPROFEN. An industrial evaluation. *J. Syst. Softw.*, 147:86–103, 2019.
18. A. K. Schömig and H. Rau. A Petri Net Approach for the Performance Analysis of Business Processes. Technical Report 116, Universität Würzburg, Germany, May 1995.
19. D. Silingas and E. Mileviciene. Refactoring BPMN Models: From 'Bad Smells' to Best Practices and Patterns. In *BPMN 2.0 Handbook*, pages 125–134. 2012.