Towards Automated Performance Optimization of BPMN Business Processes

Anastasios Gounaris

Dept. of Informatics, Aristotle University of Thessaloniki, Greece
gounaria@csd.auth.gr

Abstract. Business Process Model and Notation (BPMN) provides a standard for the design of business processes. It focuses on bridging the gap between the analysis and the technical perspectives, and aims to deliver process automation. The aim of this work is to complement this effort by transferring knowledge from the related field of data-centric workflows aiming to provide automated performance optimization of the business process execution. As a key step towards this goal, the contribution of this work is to provide a methodology to map BPMNv2.0 models to annotated directed acyclic graphs, which emphasize the volume of the tokens exchanged and are amenable to existing automated optimization algorithms. In addition, concrete examples of mappings are given, while the optimization opportunities that are opened are explained.

1 Introduction

BPMN has become an international standard for designing workflows. In principle, the basic promise of BPMN is the same diagram prepared by a business analyst to be used for automating the execution of that process on a modern process engine. This however remains a vision that rarely happens in practice and the gap between the business and the technical perspectives remains [8]. As a result, the executable workflow of business processes is either manually designed in order to provide enterprize-specific configurations or derived by simple procedures using toolkits from established vendors (e.g., Bigazi, IBM, Oracle). Either way, any performance optimization responsibilities rest with experienced IT technicians.

In this work, a different approach is advocated, according to which performance optimization is automated. First, this type of automation relieves considerable burden from workflow designers. Second, automated optimization yields intrinsically more flexible and resilient workflows. Increased flexibility stems from the fact that several equivalent alternatives are investigated by the optimizer thus providing more options. Also, when external conditions that impact on the workflow performance evolve, automatically re-optimizing the workflow is important for efficiently adapting to the new setting to attain resilience. Third, performance issues are playing an increasingly important role in modern BPM, which becomes more data- and process-intensive, e.g., in order to cope with big data [2] giving rise to the need for performance optimization. Finally, optimizers

for automatically deriving execution details is an integral component in systems that aim to allow end users to submit the process definition at a more declarative level, e.g., as discussed in [7].

Performance optimization is a field that has been largely investigated in databases (e.g., [5]) and data-centric flows (e.g., [9, 6]). Although these techniques cannot solve the problem of business process optimization in its entirety, they can form a starting point for automated performance optimization, as explained in this work. The key first step is to bridge the modeling gap: data-centric flows are typically represented as directed acyclic graphs (DAGs) and optimization techniques rely on statistical metadata, such as cost per task invocation and selectivity, which can be regarded as annotations to these DAGs. We adopt the same modeling abstraction for business processes, and we explain how BPMNv2.0 elements are translated to such annotated DAGs. The intention is to keep using the BPMN standard and the mapping to a DAG, along with the subsequent optimizations, to occur in a way transparent to the process designer automatically. In summary, the main contributions of this paper is the introduction of an annotated DAG-based approach to BPMNv2.0 modeling along with concrete examples of mappings of the main BPMNv2.0 elements, and the presentation of the optimizations enabled.[1]

In the remainder of this section, a motivating example and an overview of related work is provided. Next, we elaborate on our DAG modeling abstraction. In Sec. 3, the proposed mappings of the main BPMNv2.0 elements are provided. The optimization opportunities and the open research issues are discussed in Sec. 4 and 5, respectively.

Motivating Example. A sub-process that is encountered in banks for processing loan requests is as follows. Upon receiving a customer application, an employee fills in the applicants personal details, and then performs a series of tasks contacting trusted services from third parties on the web. Such tasks include the following: to import additional customer personal data, to check if the applicant is on any black list, to check the borrowing capacity and to check the information with the help of the national credit bureau. If any of these checks fails, the process aborts and the application is rejected. Finally, the third party services are invoked by providing the customer's SSN identity number.

This scenario is simple but capable of showing a set of optimization issues involved. We give some examples: *Which is the optimal sequence to contact the third-party services in terms of performance, given that several orderings are valid (e.g., it does not matter whether the check of the borrowing capacity precedes the check regarding the black lists and vice versa)? Should the invocations be performed in parallel? Should an employee fill full personal details only after the checks have passed?* In the envisaged approach, one can take these decisions automatically, in a principled manner in the sense that cost-based algorithms (which may well be accompanied by theoretical optimality guarantees) can be employed. Further discussion on this is deferred to Section 4.

[1] A more extended version of this work is in [3].

Related Work. Automatically devising executable workflows that speed-up execution or improve on other performance metrics is an overlooked area in business process management (BPM). Performance optimization is considered in the context of *process redesign*, which covers several topics, as discussed in [1]. Some examples are to divide an existing process into two or more separated processes, to eliminate obsolete activities, to assign tasks to the more specialized person and, in general, to perform judicious responsibility assignment, and to buffer requests to external information sources. The most relevant heuristics to database-like optimization are the so-called "business process behavior heuristics", which include *re-sequencing*, *knock-out*, and *parallelism*. Resequencing covers the optimizations that involve changing the execution order of activities, while preserving the process semantics and correctness. A specific form of re-sequencing is to move activities that check conditions, which if not met, lead to process termination, as early as possible. Such activities are termed as knock-out ones. Parallelism deals with decisions as to whether some activities should be executed in a sequential or a time-overlapping fashion. We target exactly these form of optimizations, but in a cost-based manner instead of using ad-hoc heuristics. Finally, there are techniques that restrict their optimizations in the data management tasks within business processes, e.g., [9].

2 The proposed DAG-modeled abstraction

In data-centric flows, which are also described as DAGs, each graph vertex corresponds to a task. The tasks manipulate data (e.g., extract sentiment information from tweets, combine user identifier numbers with customer info from an underlying database, and so on), and the edges denote how the transformed data flow across the tasks. Since performance in these data-centric flows is directly dependent on the volume of data being processed and the capacity of the execution engines, the optimization methodologies aim to process as fewer data as possible and make judicious assignment of tasks to resources. For the former, the key idea is to prune unnecessary data, that is data that do not contribute to the flow final desired result, as early as possible.

In business processes, the things that flow across tasks are "tokens". So, our DAGs emphasize the volume of the tokens flowing rather than the business logic and the control of the flow. Each BPMN task corresponds to a vertex in the DAG. A directed edge connects each ordered pair of vertices, between which a transmission of tokens takes place in the context of the process.

The goal of the performance optimization is to improve the average performance across multiple process executions. Performance can be crisply defined in several ways, e.g., in Section 4 we sum the costs of each activity, which reflects how efficiently tokens are processed. Statistical metadata drive the optimization procedure. This metadata are typically extracted from log files. The exact type of metadata depend on the specific optimization problem, but two types are most commonly encountered: *selectivity* and *cost* (per invocation). Selectivity is the average ratio of output to input tokens. For example, a task that, for

each given recipe, triggers a task to prepare a single meal has selectivity 1. If a task performs a check and may cause early process termination in case of the test failure, the selectivity is lower than one. Analogously, if a task produces multiple tokens per input token on average, its selectivity is above one. In all the cases, the output tokens flow across all outgoing edges of the corresponding vertex. The cost of a task is measured in the same units as the performance criterion. If performance is measured in time units, then the task cost is the average time needed to execute that task. In BPMN processes, things (captured by tasks) have to be done under certain circumstances (captured by gateways); in our DAG-based approach, the statistical metadata play an important role in considering the gateways semantics through annotations to vertices.

Apart from the statistical metadata, there are two other categories of metadata needed. The first category covers dependency constraints between task pairs, i.e., whether a task must precede another one in any execution plan to preserve semantic correctness or whether two tasks must be placed on distinct DAG branches, where each DAG branch corresponds to a different execution path. The second category captures behavioral characteristics, such as whether the task can be parallelized, so that its workload is executed by multiple executors in parallel, and whether a task can operate in a pipelined manner, i.e., to be capable of producing output tokens before consuming its entire input.

A distinctive feature of our proposal is that BPMN tasks correspond to DAG vertices but the opposite does not necessarily hold. We employ the notion of artificial tasks termed as *dummy* tasks. Overall, the combination of normal and dummy tasks with appropriately set statistical metadata allow for modeling the token flow in business processes and paves the way for performance optimization, as discussed next.

3 BPMNv2.0 Symbol Mapping

In this section, we describe how we can model the main elements of BPMNv2.0 to our annotated DAGs. The examples are deliberately simple, to convey easier our message, and they are drawn from the Camunda platform.[2] To avoid confusion, the tasks in our DAGs are depicted as circles rather than rounded rectangles. We explicitly discuss activities and gateways; [3] includes also event discussion.

Task. An ordinary task, independently of its exact type (e.g., manual, service, business rule, and so on), is mapped to a distinct vertex in our model. The cost metadata is the average cost in time units to execute that task, and its selectivity is the average ratio between the output and input tokens.

For example, in Fig. 1, there is the task *"prepare meal"*. This task is triggered after it has received the menu suggestions from the previous task. If the average time to prepare a meal is *c_pm*, then the cost of that vertex takes that value. The selectivity is set to 1, because, for each suggestion, there is a single meal prepared.

[2] `http://camunda.org/bpmn/reference/`

The loop tasks, like the *"suggest dish"* one in the figure, require a bit more attention, because they execute more than one time on average. Let us suppose that the average number of times the task is activated is n and the average time cost to execute each time is *c_sd*. There are two ways to handle this case. First, we can insert a zero cost *dummy* task before *"suggest dish"*. The selectivity of the dummy task is set to n, whereas the cost of the vertex corresponding to *"suggest dish"* remains *c_sd*. However, the selectivity of this vertex needs to become $1/n$ to account for the fact that even if n times the *"suggest dish"* task is executed, there is always one token passed on to the subsequent task. The second option is not to use a dummy task and amortize the cost of the vertex, so that it captures the fact that, on average, it is executed n times and thus becomes $n \times c\_sd$. Both options are shown in Fig. 1.

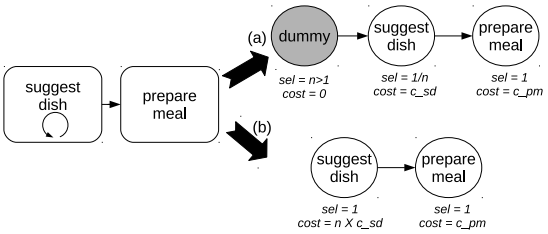

Fig. 1. Mapping a loop and an ordinary task.

In the previous examples both tasks are not parallelizable. The multiple instance tasks can be modeled in the same way as the loop ones, but the difference is that they can be parallelized up to a parallelism degree of n.

Compensation Tasks Compensation tasks can be mapped to our DAG with the help of dummy tasks as well. Consider the example in Fig. 2, where a *book trip* task with cost *c_bt* is associated with a compensating task *cancel trip* with cost *c_ct*. Let as also assume that the probability of not triggering the compensating task and continuing the normal execution is *p1*, whereas the probability of canceling the trip is *p2=1-p1*. The mapping to our DAG involves two dummy tasks, which do not contribute to the cost, but control the amount of flow to each of the two branches in a way proportional to the afore-mentioned probabilities through setting their selectivities accordingly. The preceding task in this example sends its output to both branches in line with the edge interpretation in our DAG model, and it is the responsibility of the dummy tasks to perform the filtering. The dependency constraints state that none of the two initial tasks should precede the other, i.e., the optimizer cannot place them in a sequence. Note that a simpler mapping would also be possible in cases where

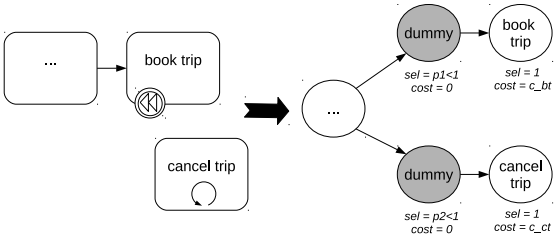

Fig. 2. Mapping a compensation task.

the two branches merge just after the book and cancel tasks. In that mapping, we could omit the dummy tasks and have only the *book trip* task with selectivity set to 1 and a weighted cost equal to $p1 \times c\_bk + p2 \times c\_ct$. This mapping does not capture the complete business logic, but is adequate for performance optimization. Similarly, if there are subsequent tasks following *book trip* but no output edge for *cancel trip*, we could have a single task with the weighted cost as above and the selectivity being equal to $p1$.

Subprocess and Call Activities. Subprocesses do not pose any specific challenge per se with regards to their mapping. However, for optimization purposes, it is always more desirable to expand them in order to broaden the optimization search space of the algorithms, provided that those algorithms are capable of navigating efficiently through the expanded search space. Also, from the performance point of view, call activities can be treated like ordinary activities in the way described above.

Adhoc. Adhoc subprocesses contain several tasks that can be executed at any order. This is exactly the sweet spot for database-like optimization, which can decide on the optimal order in a principled manner. In Fig. 3(top), we present an example with an adhoc subprocess with 4 tasks that can be executed in arbitrary order. We map them in the way shown in the right part of the figure; all the tasks in the adhoc subprocess are directly connected to the preceding task to denote that there are no inter-dependencies among them and, as such, can be computed in parallel (although the final decision rests with the optimizer as discussed later). Also, we use a zero-cost dummy combiner task to aggregate the output tokens of the adhoc tasks and call the next activity. The selectivity of that dummy task is set to 0.25 because it outputs one token for every four tokens received as input. Further, it is not pipelining, because it needs to consume all its four input tokens in each execution, before creating an output token.

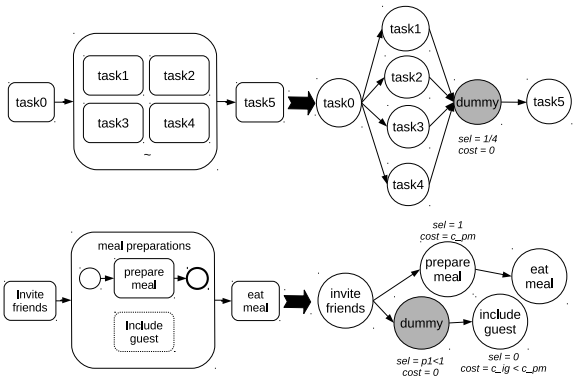

Fig. 3. Mapping an ad-hoc task (top) and an event subprocess (bottom).

Event Subprocess. An event subprocess may be executed while the enclosing subprocess is active. An example is presented in Fig. 3(bottom), where the enclosing process is a task *prepare meal*, during which new guests can be included

(captured by the event task *include guest*). The costs of these tasks are *c_pm* and *c_ig*, respectively. To map this case to our token-flow DAG, we insert a dummy filtering vertex with selectivity equal to the probability of executing the event subprocess *p1*. Note that, by definition, *c_ig* is always less than *c_pm*, and the two activities cannot be executed sequentially.

Exclusive, Parallel and Inclusive Gateways. Gateways is a core BPMNv2.0 element and a distinctive feature of process-centric flows not appearing in data-centric workflows. As such, their effective mapping is of high significance in our approach. We distinguish between exclusive, parallel and inclusive gateways. An example of an exclusive gateway is in Fig. 4, where there is an option during meal preparation for the three dishes shown. Let us assume that the average probability to select each of the three options is *p1*, *p2*, and *p3*, respectively; these probabilities need to sum to 1 to account for the fact that always exactly one option is selected. Then, we can insert zero cost dummy filtering tasks with their selectivity set equal to the probabilities above. There is also a zero-cost unary-selectivity dummy combiner task being responsible for synchronization, but this is optional. To preserve the gateway semantics, the dependencies between these vertices are that they cannot be placed in a sequence.

Now, suppose that, in the previous example, the gateway is transformed to a parallel one. Then the dummy tasks on the left become optional (corresponding to zero cost and selectivity equal to 1), so that the three previous options can be directly connected to *choose recipe*. However, the dummy task on the right becomes compulsory and its selectivity is set to 1/3, derived by a generic formula: ratio of 1 to the number of tasks after the parallel gateway. This is to ensure that *eat meal* receives a single token for each *choose recipe* execution no matter how many intermediate tasks are executed in parallel. In addition, the dummy task on the right enforces synchronization.

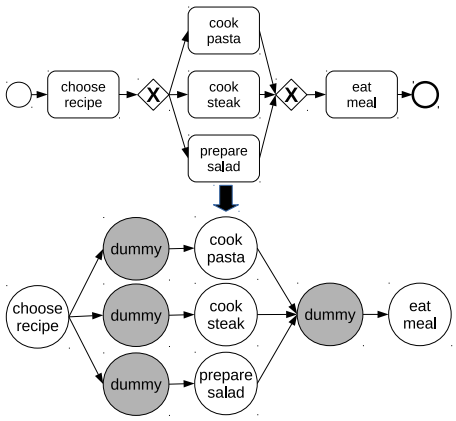

Fig. 4. Mapping an exclusive gateway.

Inclusive gateways allow tokens to flow across one, many or all paths, i.e., the combine certain features from the exclusive and parallel cases. The high-level mapping shown in Fig. 4 holds for inclusive gateways as well, but with all dummy tasks being compulsory. Contrary to the case of exclusive gateways, the selectivities of the dummy tasks preceding the options can sum to a value greater than 1. In addition, the dummy combiner task on the right part becomes

compulsory (as in parallel gateways) and its selectivity is set to the ratio of 1 to the sum of the selectivities on the left.

Gateways and Loops. Gateways are very common to be accompanied by cycles in business graph models, as shown at the top of Fig. 5. We can combine the approaches presented earlier regarding loops and gateways in order to render our graph acyclic. The tasks belonging to the loop path are placed in a sequence with their cost having been amortized as shown in Fig. 1; the alternative of a dummy task in that figure is also valid. Then, the rest of the tasks after the gateway are treated as in Fig. 4.

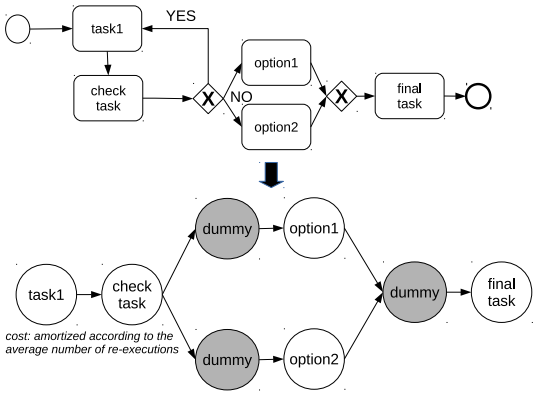

Fig. 5. Mapping a more complex exclusive gateway.

Event-based vs. Data-based Gateways In BPMNv2.0, there is a distinction between data-based and event-based gateways. The former choose the routing of a token to one or more paths according to data associated with the specific token. The latter make decisions based on events happening. From the performance point of view, there is no essential difference between these two cases. For example, consider the exclusive case. Instead of monitoring the probabilities of task activation, in an event-based exclusive gateway, we can monitor the probability of corresponding events and set the selectivities in our DAG vertices accordingly. Next, we discuss BPMNv2.0 events in detail.

4 Optimization Opportunities

The optimizer envisaged receives an initial mapping, and the set of statistical and dependency metadata and derives an optimal execution plan. A key aspect is to decide the exact order of task execution in a cost-based manner.

To provide insights into the benefits, we extend the motivating example, where the ordering of some tasks is flexible thus generalizing adhoc tasks in BPMN. Suppose a simple process, which contains n activities forming a chain (i.e., there are no branches). If, due to dependency constraints, there is no flexibility in the order, then this implies that there are $\frac{n(n-1)}{2}$ dependency edges, either explicitly stated or implied through transitive closure. For example, a loan pre-processing template may define the order in which the tasks for importing

contact information of the applicant, checking the borrowing capacity and contacting the credit bureau take place, despite the fact that any ordering is valid.

Fig.6 shows the performance improvements after simulating 100 randomly generated DAGs, where there are $0.75\frac{n(n-1)}{2}$ and $0.5\frac{n(n-1)}{2}$ dependency edges, n ranges from 10 to 15, the selectivity of tasks ranges from 0.01 (extremely filtering) to 2, and the cost ranges from 1 to 100. The value distribution is uniform. The exact optimization metric selected is the sum of the average execution time for each task. As baseline performance, which corresponds to normalized value 1, we consider the running time of the initial DAG before re-orderings. In the figure, we can see that, on average, there is a reduction in the running time by 25.62% for the more constrained case; moreover, the average reduction becomes 40.37%, for 50% constraints. Also, there are isolated runs, where the improvements can be up to an order of magnitude, as shown by the maximum improvement plots at the bottom of the figure. These numbers indicate how significant the performance benefits can be, even in simple processes. Exploring all the orderings, even in highly constrained settings, is an intractable problem. The techniques in [6] show how the optimizer can navigate through the search space in a scalable manner. Other performance optimization problems can be considered as well, as discussed in [3].

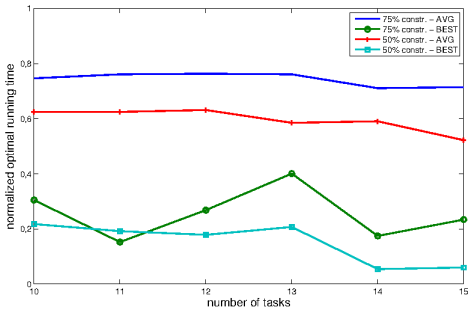

Fig. 6. Example benefits when optimally reordering activities.

5 Main Research Issues and Conclusions

Here, we mention the main research issues for developing complete solutions for performance optimization in BPMN business processes.

– *Need for dependency-aware optimization algorithms.* Mapping BPMN models to our DAG abstraction is a necessary but not sufficient condition to perform cost-based performance optimization. In the previous section, we referred to several algorithmic techniques that consider only precedence constraints, i.e., constraints of the form that task A must precede task B. This type of constraints needs to be complemented by (i) parallelism constraints that enforce tasks to be placed in different execution paths; (ii) blocking vs. pipelining information for each task; and (iii) parallelism capability information, to define which tasks are amenable to parallel execution and up to which degree of parallelism. More research is needed to develop solutions that account for the complete range of constraints in business processes.

– *Statistical Metadata Collection.* The statistical metadata play a crucial role, and their efficient collection requires special attention. Techniques like [4] may act as a starting point. Challenges include the fact that statistics are actually correlated, which means that changing the order of tasks may affect their statistical metadata.
– *Extensive Evaluation.* There needs to be extensive evaluation using benchmarks to reason with confidence about the actual capability of each proposed optimization technique.
– *Mapping to BPMN models and end-to-end solutions.* Holistic solutions should involve the mapping of the optimized execution plan back to a BPMN model and, ideally, be exposed as a software plugin to existing platforms rendering the optimization fully transparent to the process designer.

Finally, it should be remarked that, in BPM, optimization for performance only is inadequate; the focus should also be shifted to aspects such as fault-tolerance, reliability, economic cost and so on.

Summary. This work is motivated by the fact that currently, performance optimization of business process is a manual activity in the responsibility of the designer. To address this limitation, automated performance optimizations should be applied. We explain how we can build upon the knowledge in the data management community to optimize data-intensive queries and flows. More specifically, we discuss the annotated DAG modeling abstraction required to employ such solutions, going through the handling of the main BPMNv2.0 elements in detail. We provided insights into the potential performance benefits and identified the main research issues for enabling automated optimization in BPM.

References

1. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer (2013)
2. Gao, X.: Towards the next generation intelligent BPM - in the era of big data. In: Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings. pp. 4–9 (2013)
3. Gounaris, A.: Towards automated performance optimization of BPMN business processes. CoRR abs/1508.07455 (2015)
4. Halasipuram, R., Deshpande, P.M., Padmanabhan, S.: Determining essential statistics for cost based optimization of an etl workflow. In: EDBT. pp. 307–318 (2014)
5. Ioannidis, Y.E.: Query optimization. ACM Comput. Surv. 28(1), 121–123 (1996)
6. Kougka, G., Gounaris, A.: Optimization of data-intensive flows: Is it needed? is it solved? In: DOLAP. pp. 95–98 (2014)
7. Lamprecht, A.L., Naujokat, S., Schaefer, I.: Variability management beyond feature models. IEEE Computer 46(11), 48–54 (2013)
8. Stiehl, V.: Process-Driven Applications with BPMN. Springer (2014)
9. Vrhovnik, M., Schwarz, H., Suhre, O., Mitschang, B., Markl, V., Maier, A., Kraft, T.: An approach to optimize data processing in business processes. In: VLDB. pp. 615–626 (2007)