Ad-hoc subprocesses – The missing link between scheduling and business process modelling

Asvin Goel[0000−0001−6821−6535]

Kühne Logistics University, Hamburg, Germany
asvin.goel@klu.org

Abstract. This paper aims at bridging the gap between scheduling and business process modeling, two fields that have evolved independently from each other, even though both share common goals, i.e., the improvement of business operations. The paper shows how BPMN 2.0 can be used to model scheduling problems using a BPMN element that is often overlooked: ad-hoc subprocesses. It shows how ad-hoc subprocesses provide the basis for modeling scheduling requirements and demonstrates how well-known scheduling problems, such as the travelling salesperson problem and the job shop scheduling problem can be represented as BPMN models. Business problems containing scheduling problems can be directly modelled by domain experts and process owners, eliminating the need to consult experts in mathematical programming. Moreover, new avenues are opened for research in optimisation algorithms capable of operating directly on BPMN models.

Keywords: BPMN 2.0 · Modelling · Optimisation.

1 Introduction

Scheduling can be described as the process of planning and organising activities to be conducted to achieve a given purpose subject to relevant restrictions. It involves determining when and in which sequence particular activities should take place, ensuring that required resources such as time, personnel, and equipment are allocated accordingly. The main purpose of scheduling is to improve the efficiency of business operations by using mathematical optimisation techniques to eliminate idle times and bottlenecks.

Business process modelling, on the other hand, is concerned with the representation of processes and workflows, and involves a description of decision points, information flows, and the sequence in which activities shall be conducted. The main purpose of such models is to provide a structured way to understand how different parts of an organisation work together to achieve specific goals or outcomes.

Although scheduling and business process modelling are both concerned with business operations, the respective scientific fields have been evolving in isolation. There is a good reason for this development: while business process models usually provide a specific sequence in which activities are conducted, scheduling

focuses on determining the sequence in which activities shall be conducted. Thus, the respective goals appear to be contradictory.

This paper aims at building a bridge between scheduling and business process modelling with the goal of allowing the formulation of scheduling problems using business process models. This goal is achieved by leveraging a feature in the BPMN 2.0 specification that is often overlooked: ad-hoc subprocesses. The paper shows how BPMN 2.0 can be used to formulate well-known scheduling problems, like the travelling salesperson problem and the job shop scheduling problem.

2 Related work

Scheduling problems are commonly used when resources with limited capacity have to conduct multiple tasks. In such cases, a decision has to be made when which task is to be executed so that the capacity of the resource is not exceeded at any point in time. In many cases, a related decision is required, i.e., the decision when which task is to be executed by which resource. The allocation of tasks to resources has found some attention in the literature related to business process management, e.g., [2, 1, 6, 7], and a comprehensive survey is provided by [12]. Several suggestions have been made to add resource allocation information into process models. [2] and [1] present a resource assignment language and a graphical notation for resource assignments that can be used together with BPMN to provide information related to the assignment of (human) resources to tasks. Focusing on healthcare applications, [11] propose to introduce a dedicated BPMN element indicating that a task requires a resource with limited capacity. [8] propose so-called action-evolution Petri nets to model and solve dynamic task assignment problems using deep reinforcement learning. Deep reinforcement learning is also used by [9] to conduct resource allocations in business processes.

Above approaches have in common, that they assume that in order to be executed, a task needs to be assigned to one or more resources. A different perspective on resources is provided by [5], who does not model resources as entities or objects to which tasks are assigned. Instead, it is proposed to represent resources as processes or by processes managing the resource. Resource requirements can thus be represented by a collaboration between processes requiring the service of a resource and processes providing the service. Instead of including a task conducted by a resource into a process model, only a request for a service is included. The task conducted by the resource is then included in the process representing the resource, and when this task is completed, a message is sent back to the original process indicating that the resource has completed the task. Resource allocation decisions are therefore replaced by decisions regarding which request message shall be sent to which process. Compatibility constraints can easily be validated when making such decisions. To consider that most resources can only conduct one task at a time, [5] proposed so-called resource activities which require sequential execution of the tasks to be executed. As we will see, the original idea of [5] can also be applied without the introduction of resource

activities. Instead, it is possible to use BPMN ad-hoc subprocesses to ensure that a resource only conducts one task at a time.

3 Ad-hoc subprocesses

The BPMN specification [10] defines ad-hoc subprocesses as follows:

> *An ad-hoc subprocess is a specialized type of subprocess that is a group of activities that have no required sequence relationships. A set of activities can be defined for the process, but the sequence and number of performances for the activities is determined by the performers of the activities.*

The specification restricts the BPMN elements that may be used within ad-hoc subprocesses:

- **Required elements:** ad-hoc subprocesses must contain at least one activity.
- **Allowed elements:** ad-hoc subprocesses may contain data objects, sequence flows, associations, data associations, groups, message flows (as a source or target), gateways, and intermediate events.
- **Prohibited elements:** ad-hoc subprocesses must not contain start events, end events, conversations (graphically), conversation links (graphically), and choreography activities.

The specification as well as other sources explaining ad-hoc subprocesses, e.g. [3], use examples similar to the one given in Figure 1. The figure shows a subprocess with multi-instance marker ||| and adhoc-subprocess marker ∼. Several activities are included in this multi-instance ad-hoc subprocess, but not all of them are connected through sequence flows. According to the operational semantics, each activity without incoming sequence flows is enabled initially. A performer of an ad-hoc subprocess can select any of the enabled activities for execution. When any of these activities is completed, tokens are produced on each outgoing sequence flow and all tokens are forwarded as far as possible. Activities that receive a token become enabled and may be selected by the performer for execution.

Most examples known to the author describe use cases in which a certain degree of creativity or intuition is required and in which the sequence in which activities are conducted depend on some undefined behaviour of a human performer. The BPMN specification itself says that ad-hoc subprocesses do *"not contain a complete, structured BPMN diagram description"* and that they allow *"modeling of processes that are not necessarily executable"*. Camunda, a vendor of process automation software, provides the following reasoning on the use of ad-hoc subprocesses [3]:

> *Any party who executes this subprocess decides what to do and when to do it. You could say that the "barely structured" nature of what happens*

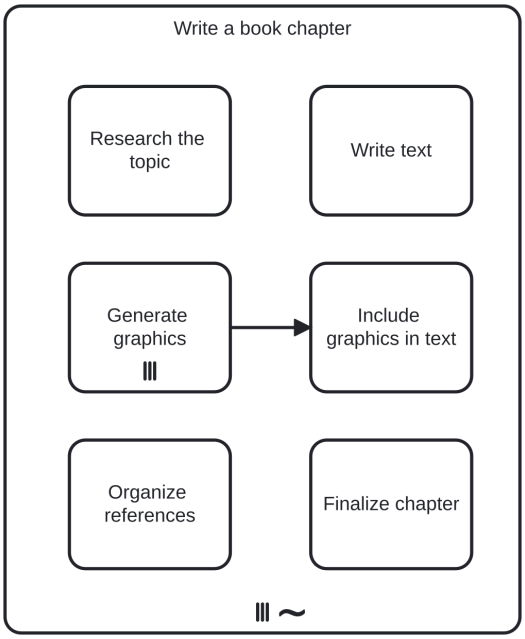

Fig. 1. An example ad-hoc subprocess

> *inside this subprocess reduces the whole idea of process modeling to an absurdity because what happens and when are the things we most want to control. On the other hand, this is the reality of many processes, and you can't model them without representing their free-form character. Frequent examples are when a process relies largely on implicit knowledge or creativity, or when different employees carry out a process differently. You can use the ad-hoc subprocess to flag what may be an undesirable actual state. Doing so could be a step on the path to a more standardized procedure.*

Given the allegedly missing structure caused by the use of ad-hoc subprocesses, they are often not perceived to be an important modelling element for many use cases. For example, the repository provided by [4] contains over 600 BPMN models, but none of them contains ad-hoc subprocesses.

A feature of ad-hoc subprocesses that is easily overlooked is the possibility to specify that activities within ad-hoc subprocesses must not be conducted in parallel by adding the attribute `ordering="sequential"` to an ad-hoc subprocess. This attribute does not influence the visual appearance of the ad-hoc subprocess and the default value is `ordering="parallel"`. By specifying that activities in an ad-hoc subprocess must be conducted in sequential order, BPMN provides the means to require that a decision has to be made which activity is conducted in which sequence. Thus, ad-hoc subprocesses with sequential ordering can be used

to model an essential feature required to model scheduling problems. BPMN does not specify how such decisions are made and states that *"performers determine when activities will start, what the next activity will be, and so on"*.

Performers are BPMN modelling elements without visual representation and define the resources responsible for conducting activities. They can be specified on process- or activity-level. A performer specified for a (sub-)process can be interpreted to be responsible for all activities within the (sub-)process. Moreover, a performer responsible for an ad-hoc subprocess is allowed to decide on how often an activity in the ad-hoc subprocess is conducted and when such an activity starts. BPMN does not restrict the behaviour of performers. Thus, it is possible to define a performer in a way that it starts each activity in an ad-hoc subprocess exactly once and freely decides on the start times of activities.

If such a performer is used for each ad-hoc subprocess with sequential ordering, we can use such ad-hoc subprocesses and performers to model scheduling problems in BPMN. Instead of using ad-hoc subprocesses for parts of a model that are lacking structure, we thus can use ad-hoc subprocess to define clearly structured models in which there is a requirement that the sequence in which activities are conducted needs to be carefully chosen to achieve operational efficiency. In general, such scheduling decisions should not be conducted at the time of modelling a process, and require knowledge of data concerning each relevant process instance.

In the following, two examples are given demonstrating how ad-hoc subprocesses with sequential ordering and respective performers can be used to model well-known scheduling problems.

4 Travelling salesperson problem

The *travelling salesperson problem* can be described as the problem of finding the shortest possible roundtrip visiting a given set of locations and returning to the origin. Each location on the roundtrip must be visited exactly once and the goal is to find the roundtrip that minimises the total distance travelled.

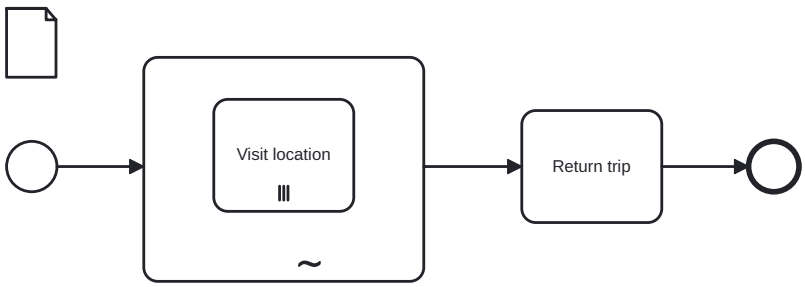

Fig. 2. The travelling salesperson problem as business process model

Figure 2 illustrates a business process model of the travelling salesperson problem. The process has a performer representing the salesperson. It starts with an ad-hoc subprocess containing a multi-instance task responsible for visiting each of the given locations. The order of visiting the locations is unspecified and, therefore, the *Visit location* task is instantiated in parallel for each location. However, as the ordering of the ad-hoc subprocess is set to sequential, the performer must not visit different locations in parallel. Thus, all locations are visited one after another in a sequence that needs to be determined by the performer. After all locations have been visited, the process continues with a task representing the return trip of the salesperson. A data object is used allowing to store the origin, the current location, and the accumulated distance of the salesperson as process variables.

To fully represent the travelling salesperson problem, the origin as well as the number of instantiations of the multi-instance activity and the respective locations must be provided as process variables. The distance to the next location can be determined using the process variables for the current location and the respective next location. The accumulated distance can be stored in another process variable to keep track of the objective value to be minimised.

An example of the travelling salesperson problem with a salesperson based in Hamburg who is required to visit Berlin, Cologne, and Munich is given in the following. Distances between the cities can be provided in a distance table as shown in Table1.

From	**To**	**Distance**
Hamburg	Berlin	296
Hamburg	Cologne	432
Hamburg	Munich	778
Cologne	Berlin	573
Cologne	Hamburg	432
Cologne	Munich	575
Berlin	Cologne	573
Berlin	Hamburg	296
Berlin	Munich	585
Munich	Cologne	575
Munich	Hamburg	778
Munich	Berlin	585

Table 1. Distance table

When the process starts, the current location is set to Hamburg and the distance travelled is set to zero. Then, an instance of the *Visit location* task is created for Berlin, Cologne, and Munich. The performer, i.e. the salesperson, now needs to decide which of these tasks to perform first. For each option, it can lookup the distance from the current location in the distance table. When the next location is selected, the current location and the total distance are updated

accordingly. After having visited all locations, the trip back to the origin, i.e. Hamburg, is conducted. If the performer always selects the location with the smallest distance, the approach resembles the nearest neighbor heuristic for the travelling salesperson problem.

5 Job shop scheduling problem

The *job shop scheduling problem* is the problem of efficiently scheduling a set of jobs on a set of machines. Each job belongs to an order, requires a certain amount of processing time on a specific machine, and must be conducted without preemption. Jobs belonging to the same order must be processed in a particular sequence. Each machine can only conduct one job at a time. The goal is to find a schedule that minimises a specific criterion, such as total completion time or makespan, while adhering to precedence relationships between jobs of the same order and the limited availability of machines.

Figure 3 illustrates a business process model of the job shop scheduling problem. The model has two processes, one describing the *Order process*, another describing the *Machine process*.

The order process starts with a multi-instance activity describing each job belonging to the order. The sequential multi-instance marker is used to indicate that these jobs must be conducted in the given sequence. Each job consists of sending a message requesting to be processed by the respective machine and waits for the machine to notify when the job is completed.

The machine process has a performer representing the machine. It contains a single task having the sole purpose of ensuring that all job requests can be received. After all job requests are received, this task is expected to terminate. Job requests are handled through a non-interrupting event-subprocess which is initiated when a job request message is received. The event-subprocess contains an ad-hoc subprocess containing a single task responsible for conducting the requested job. The performer of the machine process is responsible for all ad-hoc subprocess of all active event-subprocesses. By embedding the *Conduct job* task within an ad-hoc subprocess with sequential ordering, we can inform the performer that these tasks must not be conducted in parallel even if they belong to different event-subprocesses. After completion of a job, the event-subprocess terminates with sending a completion notification to the job. When all event-subprocesses have completed and no further job request is expected, the machine process terminates.

To fully represent the job shop scheduling problem, the number of instantiations of the *Job* activity and the respective machine required and the processing time must be provided as process variables for each order process. For each machine process, the number of jobs expected must be provided to be able to identify when the *Wait for jobs* activity can be terminated. During process execution, the number of job requests received can be stored in a process variable. A global variable can be used to capture the objective criterion, e.g., the total completion time or makespan.

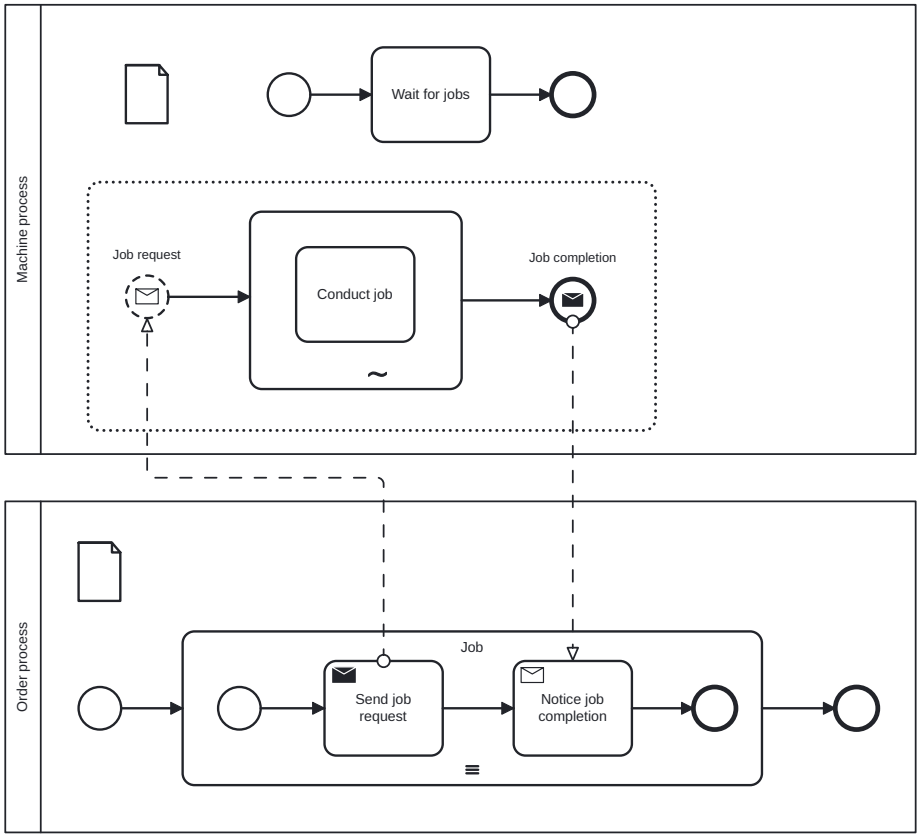

Fig. 3. The job shop scheduling problem as business process model

An example of the job shop scheduling problem with three machines and three orders is given in the following. For each order, the *Order process* is instantiated with the following data indicating the sequence of machines required and the respective processing times on these machines:

- **Order 1:** (Machine 1, 3),(Machine 2, 2), (Machine 3, 2)
- **Order 2:** (Machine 1, 2),(Machine 3, 1), (Machine 2, 4)
- **Order 3:** (Machine 2, 4),(Machine 3, 3)

For each machine, the *Machine process* is instantiated with the following data indicating the number of jobs to be conducted:

- **Machine 1:** 2
- **Machine 2:** 3
- **Machine 3:** 3

When an instance of the *Machine process* starts, the number of requests received is set to zero, the *Wait for jobs* task starts and must not be completed before the given number of jobs have been requested.

When an instance of the *Order process* starts, the *Job* activity is instantiated for each job. The first job of each order sends a request message including the required processing time to the respective machine and waits for the completion notification. Thereafter, the instantiation for the next job is executed and so on.

When a machine process receives a job request message, the respective event-subprocess starts and the required processing time of the job is stored in a local variable. The process variable representing the number of requests received is incremented. If the required number of requests have been received, the *Wait for jobs* task completes and the process terminates after all event-subprocesses have terminated.

During process execution, multiple event-subprocesses may be running for each machine. Additional event-subprocesses may be instantiated whenever a new job request is sent. Therefore, the performer of the machine process, not only has to make a decision which of the *Conduct job* tasks to conduct next, but also a decision whether to wait for new requests that could still arrive at the same or a later point in time.

Even though this example only has a few orders and machines, implementing a good decision making strategy for the job shop scheduling problem is non-trivial and underlines the necessity of developing efficient solution approaches. If such a decision making approach is available, we can use this approach to solve the job shop scheduling problem based on the BPMN model formulation provided.

6 Prototypical implementation

To provide a proof of concept, a BPMN execution engine has been designed and implemented. The execution engine reads a BPMN model in which appropriate extension elements are provided specifying data attributes, restrictions on attribute values, and operators allowing to modify data attributes. Instance data, i.e. specific values e.g. for the number of locations and their coordinates in the travelling salesperson problem, or the number of orders, jobs, and machines in the job shop scheduling problem, are read independently from the process models allowing to use the same process model for a variety of instances.

The execution engine follows the token flow logic described in the BPMN specification and advances tokens as long as no decision is required. Whenever a decision is required which activity within an ad-hoc subprocess shall be conducted next, the execution engine requests such a decision from a dedicated component, called the controller. The controller is responsible for making all decisions required during process execution and different controllers may be implemented allowing different decision mechanisms to be applied.

Preliminary tests have demonstrated the equivalency of the business process modelling formulation of selected scheduling problems with the mathematical

formulation. However, it must be noted that the quality of the solutions obtained with the prototypical implementation is not very high. At the time of writing, only basic controllers allowing to make decisions based on simple heuristic rules are implemented, e.g., a nearest neighbour heuristic for the travelling salesperson problem. It is planned to develop more sophisticated controllers based on constrained programming or deep reinforcement learning to improve the solution quality of the respective scheduling problems. However, the design of efficient controllers capable of dealing with a wide range of scheduling problems modelled as business processes is still an open research problem.

The implementation details of the execution engine and the controllers are out of scope of this paper and cannot be presented here due to page limitations. The source code including a comprehensive documentation and a collection of example models is available at `https://bpmnos.telematique.eu`.

7 Conclusion

Scheduling and business process modelling have traditionally been considered to be independent scientific fields and have evolved in isolation. While scheduling allows to improve the efficiency of business operations by eliminating idle times and bottlenecks, it usually requires experts in mathematical programming and, depending on the underlying business problem, optimisation algorithm engineers to develop dedicated approaches for solving the resulting scheduling problems. Business process modelling, on the other hand, allows domain experts and process owners without mathematical training to specify requirements of their business processes.

This paper shows how ad-hoc subprocesses with sequential ordering can be used to provide BPMN model formulations for scheduling problems and provides BPMN model examples for well-known scheduling problems. The modelling pattern allows domain experts and process owners to specify and formulate scheduling problems inherent to their use cases. Such process models can substantially facilitate the process of communicating the business problem to optimisation algorithm engineers, eliminating the need to formulate mathematical programs. Moreover, the possibility to formulate scheduling problems as BPMN models opens a new and exiting research direction of developing optimisation algorithms capable of working on a BPMN model to optimise a wide range of business problems. Simple algorithm using heuristic decision rules are already implemented, but more sophisticated decision-making approaches require further research.

References

1. Cabanillas, C., Knuplesch, D., Resinas, M., Reichert, M., Mendling, J., Ruiz-Cortés, A.: RALph: a graphical notation for resource assignments in business processes. In: International Conference on Advanced Information Systems Engineering. pp. 53–68. Springer (2015)

2. Cabanillas, C., Resinas, M., Ruiz-Cortés, A.: Ral: A high-level user-oriented resource assignment language for business processes. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) Business Process Management Workshops. pp. 50–61. Springer (2012)
3. Camunda: BPMN 2.0 Symbol Reference - All BPMN 2.0 Symbols explained with examples (2024), https://camunda.com/de/bpmn/bpmn-2-0-symbol-reference
4. Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F., et al.: Reprository: a repository platform for sharing business process models and logs. In: ITBPM@BPM. pp. 13–18 (2021)
5. Goel, A.: Towards a unifying framework for modeling, execution, simulation, and optimization of resource-aware business processes. In: Proceedings of the 2022 Winter Simulation Conference (2022). https://doi.org/10.1109/WSC57314.2022.10015245
6. Havur, G., Cabanillas, C., Mendling, J., Polleres, A.: Resource allocation with dependencies in business process management systems. In: International Conference on Business Process Management. pp. 3–19. Springer (2016)
7. Ihde, S., Pufahl, L., Völker, M., Goel, A., Weske, M.: A framework for modeling and executing task-specific resource allocations in business processes. Computing (2022). https://doi.org/10.1007/s00607-022-01093-2
8. Lo Bianco, R., Dijkman, R., Nuijten, W., van Jaarsveld, W.: Action-evolution Petri nets: A framework for modeling and solving dynamic task assignment problems. In: Di Francescomarino, C., Burattin, A., Janiesch, C., Sadiq, S. (eds.) Business Process Management. pp. 216–231. Springer Nature Switzerland, Cham (2023)
9. Middelhuis, J., Bianco, R.L., Scherzer, E., Bukhsh, Z.A., Adan, I.J.B.F., Dijkman, R.M.: Learning policies for resource allocation in business processes (2024)
10. Object Management Group: Business Process Model and Notation (BPMN) 2.0.2 (2013), http://www.omg.org/spec/BPMN/2.0.2/PDF
11. Onggo, B.S.S., Proudlove, N., D'Ambrogio, S., Calabrese, A., Bisogno, S., Levialdi Ghiron, N.: A BPMN extension to support discrete-event simulation for healthcare applications: an explicit representation of queues, attributes and data-driven decision points. Journal of the Operational Research Society 69(5), 788–802 (2018)
12. Pufahl, L., Ihde, S., Stiehle, F., Weske, M., Weber, I.: Automatic resource allocation in business processes: A systematic literature survey. arXiv preprint arXiv:2107.07264 (2021)