WEASY: A Tool for Modelling Optimised BPMN Processes

Angel Contreras, Yliès Falcone, Gwen Salaün, and Ahang Zuo

Univ. Grenoble Alpes, CNRS, Grenoble INP, Inria, LIG, 38000 Grenoble France

Abstract. Business Process Model and Notation (BPMN) is a standard modelling language for workflow-based processes. Building an optimised process with this language is not easy for non-expert users due to the lack of support at design time. This paper presents a lightweight modelling tool to support such users in building optimised processes. First, the user defines the tasks involved in the process and possibly gives a partial order between tasks. The tool then generates an abstract graph, which serves as a simplified version of the process being specified. Next, the user can refine this graph using the minimum and maximum execution time of the whole graph computed by the tool. Once the user is satisfied with a specific abstract graph, the tool synthesises a BPMN process corresponding to that graph. Our tool is called WEASY and is available as an open-source web application.

1 Introduction

The Business Process Model and Notation [5] (BPMN) is a workflow-based notation published as an ISO standard. BPMN is currently the popular language for business process modelling. However, specifying processes with BPMN is difficult for non-experts and remains a barrier to the wide adoption of BPMN in the industry. While process mining techniques [7] help to infer processes from execution logs automatically, they do not provide a solution to make users more comfortable with BPMN. Another solution presented in [6] aims at converting text to BPMN, but the text considered as input is not user-friendly and must respect a precise grammar. Moreover, optimization is a tricky phase, which should be considered as soon as possible during the development process. Early integration of optimization allows for building efficient processes, hence reducing the process execution time and the associated costs.

In this paper, we propose a semi-automated approach for supporting users in the modelling of business processes to build optimised BPMN processes at design time. The main idea is to start with a rough model of the process-to-be and refine it by introducing further details in the process step by step. More precisely, as a first step, we expect from the user that (s)he defines the set of tasks involved in the process. Each task comes with a range of minimum and maximum durations. Given a set of tasks and a partial order between some of these tasks, the tool automatically generates an abstract graph, which serves as a first version of the process. We use an abstract graph for modelling purposes because it avoids introducing gateways and possible complex combinations of gateways necessary for expressing looping behaviour, for example. Given such an abstract graph, the tool can compute the minimum and maximum times for executing the whole graph. This information regarding time analysis is particularly interesting for optimization purposes and the user can rely on this information for refining the abstract graph, particularly by reducing this execution time and thus the associated costs. When the user is satisfied with a specific abstract graph and its corresponding execution times, (s)he can decide to generate the corresponding BPMN process automatically.

All these features are implemented in Python and integrated into a web application, which serves as a front-end UI where the user can call these functionalities and visualise the results. The tool, called WEASY[1], was applied to several case studies for evaluation purposes and results were satisfactory in terms of usability and performance.

The rest of the paper is organised as follows. Section 2 introduces BPMN and other models used in this work. Section 3 surveys WEASY's functionalities. Section 4 presents some experimental results. Section 5 illustrates how our tool works on a case study. Section 6 concludes.

2 Models

In this work, we consider a subset of BPMN [5] focusing on behavioural aspects (including start/end events, tasks, flows, exclusive/parallel gateways) and time (task duration). As far as time is concerned, each task defines a range of durations indicating the minimum and maximum duration it takes to execute that task. Once the task completes, its outgoing flow is triggered.

WEASY relies on a notion of partial order between tasks. A partial order consists of a set of tasks (all tasks involved in a process) and a relation between tasks. When two tasks are related, it means that the first one must execute before the second one in the final process. This notion of partial order is used to simplify the modelling of BPMN processes.

Abstract graphs are primary models of WEASY, and serve as abstract representations of BPMN processes. An abstract graph consists of a set of nodes and a set of directed edges. A node is defined as a set of tasks and a set of graphs. A graph is thus a hierarchical structure. Abstract graphs are simpler than BPMN processes because they do not rely on nodes and gateways, which can yield intricate structures when they are nested or express specific behaviours such as loops or unbalanced structures. Note that nodes are defined using parallel execution semantics. This means that all the tasks and graphs in a node execute in parallel. A choice in the graph can be expressed by several edges outgoing from the same node. Looping behaviour can be expressed using an edge going back to a predecessor node.

3 Tool

The WEASY tool consists of two parts. The first part of the tool implements in Python the main features of the approach: (i) transformation from a partial order of tasks to

[1] The source code and the instructions for installing and using WEASY are available at: `https://github.com/ahzm/weasy`

an abstract graph, (ii) computation of the minimum/maximum execution time of an abstract graph, and (iii) transformation of an abstract graph to BPMN. More precisely, the user first gives as input a set of tasks and a partial order of tasks, then Algorithm (i) processes this set of inputs and returns a hierarchical abstract graph. Algorithm (ii) goes through this abstract graph and computes its minimum and maximum execution times. Next, the user decides whether to refine this abstract graph by adjusting the position of nodes (tasks) based on this returned result. Finally, Algorithm (iii) transforms the abstract graph into a standard BPMN model. The reader interested in more details regarding the algorithms behind these features should refer to [3]. Figure 1 shows the classes implemented in Python. These classes encode the different models used in our approach (partial order, abstract graph, BPMN graph) as well as the algorithms used for automating the main steps of the proposed modelling technique.

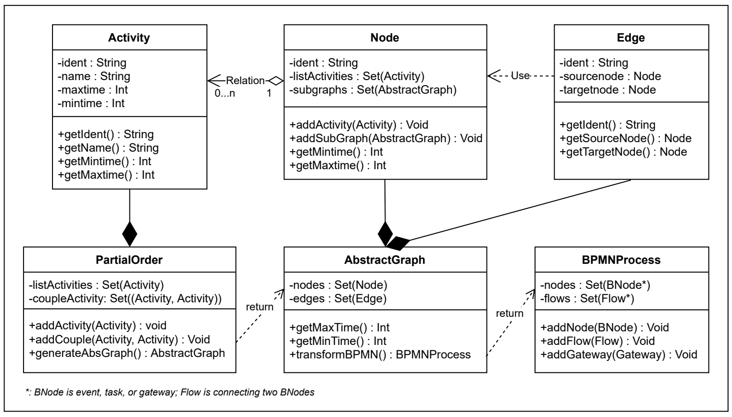


Fig. 1. Class diagram of the tool features implemented in Python.

The second part of the tool corresponds to a web application or User Interface, which can be used to design a process by defining a partial order, refining abstract graphs and finally generating the BPMN process. All the algorithms implemented in Python can be called using buttons, and all the results (abstract graphs, BPMN) can be visualised in the web application. This web application was implemented by using the following JavaScript libraries: *React*, *MxGraph* and *bpmn-js*.

4 Experiments

In this section, we report on experiments to evaluate performance of the tool and usability of the approach with respect to manual modelling.

Performance. We evaluated the performance of the three main features of our implementation, in practice, by applying the tool to input models of increasing size. These experiments were run on a Windows 10 machine with a Core i5@1.70Ghz processor and $16\,\mathrm{GB}$ of RAM. Table 1 presents the execution time of the algorithm generating an abstract graph from a partial order. We vary the number of tasks (first column) and the size of the partial order given as input. The second column gives the percentage of tasks related to another one according to the partial order. For instance, for 1000 tasks, 90% means that 900 tasks (out of 1000) are used in the partial order relation set. The last column shows the time required to execute the algorithm. When increasing the number of tasks (up to 1000), the time remains reasonable (less than 0.5 second). For very large applications involving $10\,000$ tasks, we can note the different execution times for generating the initial abstract graph. When the total number of tasks remains the same and the size of the partial orders increases, then its execution time decreases. This stems from the fact that the constraints on the graph to be generated augment with the number of tasks in the partial order. The more constraints on the graph, the less computation is required. For example, when the proportion of tasks appearing in the partial order is 30%, the execution time takes about 28 seconds. Conversely, when the percentage of tasks is 90%, the execution takes about 4 seconds.

Table 1. Execution time of the algorithm converting a partial order to abstract graphs.

# Tasks	Partial Order	Time (s)
10	90%	≈ 0
100	90%	≈ 0
500	90%	0.02
1000	90%	0.05
10 000	30%	27.76
	60%	15.36
	90%	4.34

Table 2 presents the time for computing the minimum/maximum time according to different abstract graphs. The table contains five columns exhibiting the graph identifier, the number of nodes, the number of edges, the number of tasks, and the algorithm execution time. The execution time is less than a second for graphs containing up to 300 nodes and edges. However, for larger graphs with 500 nodes or more, it takes more than a second (about 16 seconds for the largest example with 1000 nodes in the table). This increase mainly comes from the graph size, that is, the number of nodes and edges that all need to be traversed by this algorithm. Execution time also depends on the structure of the graph. For instance, note that for graphs $G5$ and $G6$, it takes more time to execute the algorithm for 100 nodes ($G5$) than for 300 nodes ($G6$), because $G5$ exhibits more paths to be explored (due to interleaving and loops) than $G6$.

The algorithm transforming abstract graphs to BPMN processes is very efficient (less than a second for all the entries in Table 2).

To summarise, it is worth noting that, even if we have shown examples with hundreds or thousands of tasks, nodes, and flows/edges in this section, in practice, BPMN

Table 2. Execution time for the minimum/maximum time computation.

Identifier	Nodes	Flows	Tasks	Time (s)
$G1$	5	4	10	≈ 0
$G2$	30	37	50	≈ 0
$G3$	30	38	100	≈ 0
$G4$	50	63		0.01
$G5$	100	120	500	0.62
$G6$	300	313		0.11
$G7$	500	524	1000	13.81
$G8$	1000	1020	10 000	16.08

Table 3. Empirical study results.

Group	Correctness	Max time	Modelling time
Manual	50%	32 days	33 min.
Semi-auto.	100%	30 days	10 min.

processes/graphs are usually rather small (less than 100 nodes). For this size, all our algorithms are very efficient (less than a second).

Usability. We carried out an empirical study to assess the usefulness of our approach compared to classic manual modelling. We have provided two groups of three people (all being non-expert users) with an informal description of a business process (an employee hiring process in a company, see Section 5). This description makes explicit the list of activities (with minimum/maximum duration) and the set of ordering constraints to be respected by some of these activities. The goal of the exercise was to provide a BPMN model corresponding to this problem. The BPMN process should (i) use all tasks, (ii) satisfy all ordering constraints, and (iii) be as optimised as possible, that is, the maximum execution time of the process should be as short as possible. The first group of people did not use any tool to support the modelling phase, whereas the second group used the approach and tool proposed in this paper. It was also clearly stated in the exercise that several successive versions of the process could be built, but only the final version was sent as the result.

We have evaluated the results provided by each person using three criteria: correctness of the result, maximum time of the final process, and modelling time. As far as the correctness is concerned, this criterion checks whether the model satisfies the ordering constraints. It is worth noting that, for the given problem, a single process was correct with respect to the given ordering constraints. The second criterion is the maximum execution of the process, which must be as short as possible (the employee hiring process lasts a maximum of 30 days in its most optimised version). Modelling time is the time the person takes to build the final BPMN process. Table 3 details the results for each group of people. Each line in the table presents the average of the results for each group.

Interestingly, the quality of the model (correctness with respect to initial requirements) is much lower using manual modelling, which is indeed more prone to mistakes. The maximum execution time of the resulting process is slightly higher using manual modelling, showing that optimisation is more difficult without using a tool systematically computing this time. Finally, it is much longer (more than three times) to obtain the proposed final process compared to our semi-automated approach.

5 Case Study

We illustrate how our tool works with an employee recruitment process. This process focuses on the different tasks to be carried out once the employee has successfully passed the interview. The employee has to complete some paperwork. (S)He has to see the doctor for a medical check-up. If the employee needs a visa, (s)he should also apply for a work visa. At some point, (s)he should submit all documents. If these documents are unsatisfactory, the company may ask for them again. If everything is fine, all documents are accepted as is. All provided documents are archived properly once validated. The employee is also added to the personnel database and Human Resources (HR) anticipate wage payment while an assistant prepares the welcome kit (office, badge, keys, etc.).

According to this short description of the expected process, the user (someone from the HR staff for example) first needs to define the corresponding tasks and gives an approximate duration for each task as follows:

- T_1: "Fill-in form", $[1$ day - 2 days$]$
- T_2: "Medical check-up", $[1$ day - 5 days$]$
- T_3: "Visa application", $[7$ days - 14 days$]$
- T_4: "Submit documents", $[1$ day - 2 days$]$
- T_5: "Documents accepted", $[1$ day - 2 days$]$
- T_6: "Documents rejected", $[1$ day - 2 days$]$
- T_7: "Archive all documents", $[1$ day - 3 days$]$
- T_8: "Update personnel database", $[1$ day - 2 days$]$
- T_9: "Anticipate wages", $[3$ days - 10 days$]$
- T_{10}: "Prepare welcome kit", $[3$ days - 5 days$]$

The user can then define an order between some of these tasks. In the case of this example, the following ordering constraints are defined by the user:

- submitting documents can only appear after filling forms, medical check-up, and visa application $\rightsquigarrow$ (T_1, T_4), (T_2, T_4), and (T_3, T_4);
- documents are accepted or rejected once they have been submitted $\rightsquigarrow$ (T_4, T_5) and (T_4, T_6);
- archiving documents, updating database, anticipating wages, and preparing welcome kit can appear only after validation of the documents $\rightsquigarrow$ (T_5, T_7), (T_5, T_8), (T_5, T_9), and (T_5, T_{10});
- updating personnel database should be executed before anticipating wages and preparing welcome kit $\rightsquigarrow$ (T_8, T_9) and (T_8, T_{10}).

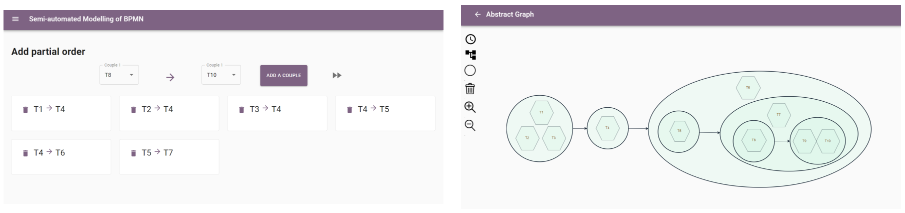
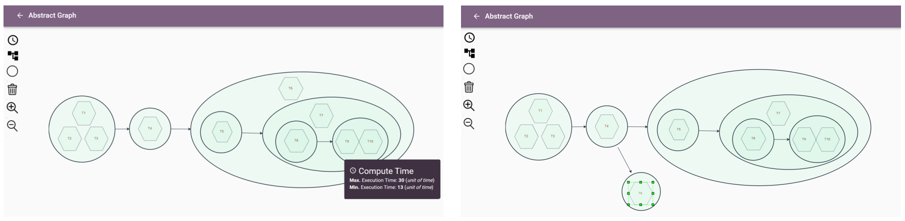


Fig. 2. Definition of partial orders using WEASY (left), and generation and visualization of the first graph (right).

The definition of tasks and ordering constraints is achieved using WEASY on dedicated interfaces, as illustrated in Figure 2 (left).

The tool then takes this data (set of tasks and set of pairs) as input, and returns the abstract graph given in Figure 2 (right). The generation algorithm works as follows. First, it detects that tasks T_1, T_2, and T_3, have as common successor task T_4. Then T_4 is the shared task, splitting the directed graph into two parts. Therefore, the algorithm creates a node to store T_1, T_2, and T_3 and a second node to store T_4. Since T_4 has two successor tasks T_5 and T_6, the algorithm then creates a new node to store them. Since T_5 has some successor tasks, and T_6 does not have any successor task, the algorithm creates a subgraph in this new node and moves T_5 to this subgraph. The rest of the algorithm execution extends this subgraph to integrate the remaining tasks and finally returns the graph given in Figure 2 (right).

We can then click in the menu on the clock icon (left top corner) in order to compute execution times for this graph. As shown in Figure 3 (left), the tool returns 13 as minimum execution time and 30 as maximum execution time. This abstract graph contains three nodes. The algorithm first computes the minimum/maximum execution time of the nodes. These times for the first node are 7 and 14, 1 and 2 for the second node, and 5 and 14 for the last node. Since the nodes in the graph are executed sequentially, the execution times of these three nodes are summed to compute the final result.



Max. Execution Time: 30 (unit of time)
Min. Execution Time: 13 (unit of time)

Fig. 3. Computation of times for the first graph (left), and refinement of the graph by moving a task (right).

In most cases, the initial graph can be improved, and this is the goal of the refinement steps. Consider the abstract graph shown in Figure 3 (left). For this abstract graph, the refinement process consists of two steps. Tasks T_5 and T_6 (documents accepted or rejected) should appear in two different nodes. To do so, in the first step, we add a new node to the graph. In a second step, we move the task T_6 to this new node as shown in Figure 3 (right). We then recompute the execution times of this new abstract graph. Its minimum execution time is 9 and its maximum execution time is 30. After the refinement step, the maximum time is the same, but the minimum time has improved (going from 13 to 9).

Finally, from the abstract graph given in Figure 3 (right), we call the BPMN generation algorithm for obtaining the resulting BPMN process shown in Figure 4. We can see that the first node with three tasks (T_1, T_2, T_3) transforms into a split and a join parallel gateway. The algorithm also generates a split exclusive gateway right after T_4 because the corresponding node has two outgoing edges. After T_5, there are again several parallel gateways because there are multiple tasks within the same node.

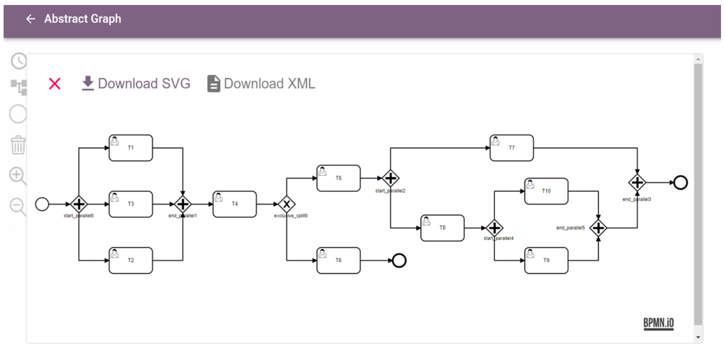


Fig. 4. Transformation of the final graph to BPMN.

6 Concluding Remarks

In this paper, we have presented a tool that facilitates the modelling of optimised BPMN processes. Our solution relies on a simple notation for describing an abstract version of a process called an abstract graph. An abstract graph is first generated by defining the set of tasks and a partial order between some of them. The user can then successively refine this abstract graph. Refinement is guided by the minimum and maximum execution times needed for executing the whole process. Once the user is satisfied, the last step transforms this graph into a BPMN process. All the steps of our approach have been implemented in the WEASY tool, which was successfully applied to a series of examples for validation purposes. The main perspective of this work aims at enlarging the considered BPMN subset to take additional constructs into account, such as data, conditions or probabilities in exclusive gateways, see [1,2,4] for instance.

Acknowledgements. This work was supported by the Région Auvergne-Rhône-Alpes within the ”*Pack Ambition Recherche*” programme.

References

1. F. Durán, C. Rocha, and G. Salaün. Symbolic Specification and Verification of Data-Aware BPMN Processes Using Rewriting Modulo SMT. In *Proc. of WRLA'18*, volume 11152 of *LNCS*, pages 76–97. Springer, 2018.
2. F. Durán, C. Rocha, and G. Salaün. A Rewriting Logic Approach to Resource Allocation Analysis in Business Process Models. *Sci. Comput. Program.*, 183, 2019.
3. Y. Falcone, G. Salaün, and A. Zuo. Semi-automated Modelling of Optimized BPMN Processes. In *Proc. of SCC'21*. IEEE, 2021.
4. Y. Falcone, G. Salaün, and A. Zuo. Probabilistic Model Checking of BPMN Processes at Runtime. In *Proc. of IFM'22*, volume 13274 of *LNCS*, pages 191–208. Springer, 2022.
5. ISO/IEC. International Standard 19510, Information technology – Business Process Model and Notation. 2013.
6. 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.
7. W. M. P. van der Aalst. Process Mining. *Commun. ACM*, 55(8):76–83, 2012.