# Quantifying the Similarity of BPMN Processes

Gwen Salaün

# Quantifying the Similarity of BPMN Processes

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

***Abstract*—Business Process Model and Notation (BPMN) is a graphical modelling language for specifying business processes. Among the open issues existing in the business process development, one of them aims at providing techniques for comparing two versions of a process model. Comparing processes is useful for tackling several problems such as process reconfiguration or evolution, process harmonization or effective search. In this paper, we propose two measures of similarity between two versions of a BPMN process. The first one relies on the syntactic descriptions of the two processes considered as input, whereas the second one focuses on their semantic models. These two measures are complementary and allows users to better understand the differences and similarities between the two processes. Our approach is fully automated by several tools we reused or implemented for this work.**

***Index Terms*—Business Processes, BPMN, Automated Verification, Quantitative Analysis.**

## I. Introduction

Business Process Management (BPM) is the activity of discovering, representing, improving and optimizing processes of an enterprise, so that these processes can be better understood and controlled. Business Process Model and Notation (BPMN) is a workflow-based graphical notation for specifying business processes. The latest version is BPMN 2.0 and was published as an ISO standard a few years ago [1]. In this context, one existing challenge is to provide techniques for comparing two versions of a process model. Although this is rather simple to detect that two processes are the same or different, this is much more difficult, when they are different, to quantify their degree of similarity. Measuring the differences between two processes is very useful for tackling several problems such as process reconfiguration or evolution [2], process harmonization [3] or effective search [4]. More precisely, as an example, when one makes a process evolve, by refactoring it or by adding new features in it for instance, it is important to be able to check whether, and how, this process has changed, and possibly correct flaws introduced during the evolution process. Therefore, there is a need for automated techniques that would help the designer to understand the similarities and differences between a current process and a new version of it.

In this paper, we propose to quantify the distance between two versions of a process modelled with BPMN. We first compare the two BPMN processes from a syntactic perspective, thus providing a first measure of comparison. In addition, we propose a second measure, which focuses on the behavioural semantics of both processes. This semantic measure works in two steps. First, we transform each BPMN process into process algebra, which allows us to generate a Labelled Transition System (LTS) representing all possible executions of the BPMN process. LTS is used as a semantic model for BPMN. Second, we use a measure of comparison between two LTSs. This measure returns a detailed quantification of the distance between all the states involved in both LTSs, as well as a global measure indicating how far both LTSs are one from another. In order to have a correspondence between the semantic model (LTS) and the initial syntactic process, we also compute a mapping between the states in the LTS and the nodes in the BPMN process. This mapping is particularly useful to better understand what parts of the syntactic process are different according to the semantic measure. As far as tool support is concerned, the approach presented in this paper relies on existing tools and new tools we have implemented in order to automate the whole computation of the similarity measures. We have applied this approach to several realistic BPMN processes for validation purposes and the results are satisfactory in terms of performance and accuracy.

The organization of the rest of this paper is as follows. Section II introduces the BPMN subset considered in this work. Section III presents the similarity measure of two syntactic BPMN models. Section IV presents the similarity measure of two semantic BPMN models, which relies on a transformation to LTS and on LTS comparison. Section V describes the tool support and evaluation of the approach. Section VI compares our solution to related work. Section VII concludes the paper.

## II. BPMN

BPMN is a graphical notation for modelling business processes as collections of related tasks that produce specific services or products for particular clients. BPMN is an ISO/IEC standard [1], and can be executed by using an existing business process platform (*e.g.*, Activiti, Bonita BPM, Camunda, etc.). In this paper, our goal is not to consider the whole expressiveness of the BPMN language, but to concentrate on the BPMN elements related to control-flow modelling, which are always used to model BPMN processes. Specifically, we consider start/end events, tasks, gateways and sequence flows. 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 gateway is used to control the divergence and convergence of the execution flow. A sequence flow describes two nodes executed one after the other.

Gateways are crucial since they are used to model control flow branching in BPMN and therefore influence the overall process execution. We consider the following gateway types: exclusive, inclusive, parallel and event-based. Gateways with one incoming branch and multiple outgoing branches are called *splits*, *e.g.,* split inclusive 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. An inclusive gateway executes any number of branches among all its incoming or outgoing branches. A parallel gateway creates concurrent flows for all its outgoing branches or synchronizes concurrent flows for all its incoming branches. An event-based gateway takes one of its outgoing branches or accepts one of its incoming branches based on events. In this work, we support unbalanced workflows and looping behaviours.

**Running example.** Figure 1 shows an online evisa service described as a BPMN process. The applicant has to fulfil several online forms and upload a scanned version of the passport. The application is then evaluated and a result is notified to the applicant (accept or reject). In case of acceptance, the applicant has to pay for fees, the bureau in charge of visa delivery prepares the requested document, and an electronic version of the visa is delivered by email.

## III. Syntactic Similarity

In this section, we first present an abstract graph model for describing BPMN processes. This BPMN graph is used in a second step for defining a syntactic measure of comparison between two BPMN graphs. We finally illustrate this syntactic measure on the running example.

A graph consists of a set of nodes and a set of edges connecting these nodes. A node is either a start node, an end node, an activity, a split gateway or a join gateway. Note that a split or join gateway is of one of the supported types (exclusive, inclusive, parallel, event-based).

*Definition 1 (BPMN Graph):* A BPMN graph is a directed graph $G = (N, E)$ where $N$ is a set of BPMN nodes and $E$ is a set of directed edges, each edge connecting two nodes. A BPMN node is either a start/end node, a task node, a split node or a merge node. In case of split and merge node, the node also comes with a type which is either exclusive, parallel, inclusive or event-based.

Given two BPMN graphs, we propose a measure comparing all nodes according to several criteria, which exploit all elements present in both graphs. For each couple of nodes, we compare them with respect to some local criteria, such as the node types or the number of incoming/outgoing flows, and some global criteria which analyse the positions of these nodes in the whole BPMN graph. For each couple of nodes (one belonging to each BPMN graph), we compute a degree of similarity which belongs to [0..1]. All these results are stored in a matrix where nodes of one graph appear in row and nodes of the other graph appear in column.

As far as local criteria are concerned, we compare two nodes $n_1 \in G_1$ and $n_2 \in G_2$ by comparing the following characteristics: node types, number of incoming flows, number of outgoing flows and similarity of neighbour nodes. More precisely, if the two nodes are both start/end nodes, this is a perfect match. If both nodes are activities, we also compare the name of the activity. If activity names are the same, this is a perfect match. If both activities have different names, we return 0.5. If the two nodes are splits or merges, this is at least 0.5. For this kind of nodes, we also add 0.25 if the number of incoming (outgoing, respectively) flows is the same. If the two nodes have different types, we return 0 for this measure.

The last local criterion focuses on the similarity of neighbour nodes (all of them). The neighbours of a node are all its successor and predecessor nodes. More precisely, given two nodes $n_1$ and $n_2$, this similarity measure is obtained by comparing the predecessors of both nodes, by comparing the successors of both nodes, and by computing the average of all these similarity measures.

The two global criteria aim at comparing two nodes $n_1 \in G_1$ and $n_2 \in G_2$ by looking at their positions in their respective graphs. We rely on two measures: (i) comparison of distance from initial nodes to nodes $n_1$ and $n_2$, (ii) comparison of distance between $n_1$ and $n_2$ to the closest end node. In both cases, we search for the shortest path, which gives an estimation of the position in the graph while avoiding to analyse all possible paths outgoing of (leading to, respectively) these nodes. Both measures are then computed in the same way as follows: $1 - (abs(d_1 - d_2)/max(d_1, d_2))$, where $d_1$, $d_2$ is the distance from $n_1$, $n_2$ to the closest end node or from initial nodes to $n_1$, $n_2$. Function $abs$ is the absolute value function and $max$ returns the longest distance. Intuitively, this formula returns a high value for nodes which are at the same distance, and lower values if this distance increases.

For each couple of nodes (one from each BPMN graph), the resulting value in the matrix is computed by using the weighted average (*e.g.*, equal or arbitrary weights) of all the aforementioned values. Note that these weights are parameters of our approach. We can decide to change them for putting more emphasis on node comparison or global criteria for instance. In the results presented in this paper, we chose equal weights for these parameters.

Since the similarity of neighbour nodes uses the matrix itself, we use an iterative algorithm that stops when the matrix stabilizes. In practice, the iterative process terminates when the distance $\delta$ between two versions of the matrix goes below a fixed threshold. The distance between two matrices is obtained by computing the arithmetic mean of the difference of the two same nodes in each matrix. Note that the computation of the matrix always converges to a unique similarity matrix. This convergence can be proven as achieved in [5] by using Banach's fixed point theorem.

Finally, we can rely on the matrix for computing a global value of syntactic similarity. We present two possible options (there are more). The first one computes the average of the best scores for each row and for each column. This option adopts an optimistic point of view and makes an attempt at matching nodes with high scores. The second option computes the average of scores above a threshold (e.g., 0.8), assuming all these values are relevant since they identify similar nodes

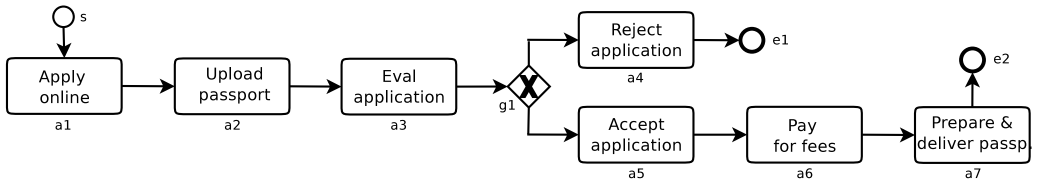


Figure 1. Evisa Application Process (V1)

in both processes.

**Example.** The initial version of the BPMN process for the online evisa service was rather simple. We improve it and the result is shown in Figure 2. This new version takes a new task into account, which checks whether the quality of the scanned passport is good enough. If this is not the case, the applicant is asked to upload again its passport. If the quality is satisfactory, the application is then evaluated. The second improvement concerns the preparation and delivery of the passport. This task is now split in two, and the preparation of the passport is achieved in parallel of the payment of fees by the applicant. This change can be seen like an optimization of the final steps of the initial process to reduce the overall execution time of the whole process.

Table I shows the resulting matrix for the syntactic similarity measure (values greater than 0.8 are in red, and values between 0.6 and 0.8 are in violet). This table first shows that similar parts in both processes have high values. This is the case for instance of a1 in V1 and a1 in V2 (0.84), a4 in V1 and a6 in V2 (0.87), or g1 in V1 and g3 in V2 (0.81). We can also see that the two new parts in V2 with respect to V1 do not have clear counterparts in V1. For example, the new task a3 in V2 (check quality) does not exhibit a high value with any task of V1 (highest value is 0.64). Similarly, the new parallel gateways (g4 and g5) do not have high values with any node from V1. This shows that the syntactic similarity results are quite accurate by returning high values for similar nodes and lower values for new nodes without any clear counterpart in the other process.

Global measures of syntactic similarity return 77% (average of best scores) and 85% (average of best scores above threshold of 0.8), respectively, which is normal since both processes are rather similar. 85% is quite high, but this is because we keep only values above 0.8. If we reduce this threshold to 0.7, the global measure of syntactic similarity decreases to 79%.

To sum up, the global measure gives a unique value of similarity between the two processes (how far they are one from another from a syntactic point of view), whereas the matrix gives detailed values showing which nodes are similar or not. The matrix is particularly useful if two processes are rather complicated. In that case, pointing out similarities and differences manually is difficult. The values in the matrix allow one to identify these similar (different, resp.) nodes.

## IV. Semantic Similarity

This section presents the semantic similarity measure, which relies on a transformation from BPMN to Labelled Transition System (LTS), and on a similarity measure between LTSs. The semantic similarity is complementary to the syntactic one. For example, two processes can be different from a syntactic point of view but can have identical semantic models. As another example, a slight difference in one process (e.g., replacing an exclusive split with a parallel split gateway) can result in a completely different semantic model. Therefore, the semantic similarity measure gives different information compared to the syntactic one by focusing on the semantic models and thus highlighting common parts and differences in these models. In this work, we preferred a semantic model for BPMN based on LTSs (instead of using Petri nets for instance), because there are well-known techniques in the concurrency theory to compare LTSs (such as simulation preorders or bisimulations [6]). This section also defines a mapping between the syntactic BPMN graph and its semantic model, which helps to identify the similar and different parts from a semantic perspective on the original BPMN model.

### *A. BPMN to LTS*

We present in this section an overview of the model transformation from BPMN to LTS, obtained through a transformation from BPMN to the LNT process algebra. We chose to transform BPMN to LNT because they both share similar constructs (e.g., choice or parallel composition) and LNT is equipped with an LTS semantics.

LNT [7] is an extension of LOTOS, an ISO standardized process algebra [8], which allows the definition of data types, functions, and processes. LNT processes are built from actions, choices (select), parallel composition (par), looping behaviours (loop), conditions (if), and sequential composition (;). The communication between the process participants is carried out by rendezvous on a list of synchronized actions included in the parallel composition (par). Thanks to the LTS semantics of process algebras, one can use thereafter the rich set of tools existing for LTS-based verification. We use in this work the CADP toolbox [9], which takes as input an LNT specification and generates the corresponding LTS.

The main idea of this transformation is to encode as LNT processes all BPMN elements involved in a process definition, that is, the nodes (start/end, tasks, gateways), which correspond to the behaviour of the BPMN process, and sequence flows, which encode the execution semantics of the process.

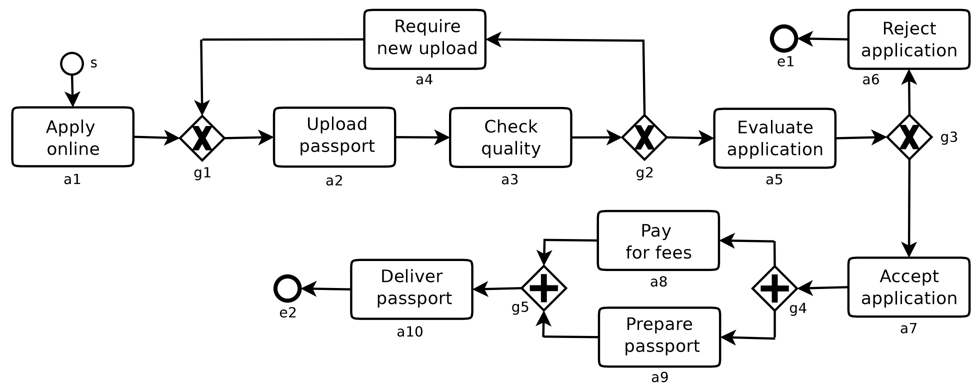


Figure 2. Evisa application process (V2)

| | a1 | a10 | a2 | a3 | a4 | a5 | a6 | a7 | a8 | a9 | e1 | e2 | g1 | g2 | g3 | g4 | g5 | s |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a1 | 0.84 | 0.16 | 0.53 | 0.55 | 0.43 | 0.39 | 0.24 | 0.19 | 0.17 | 0.17 | 0.04 | 0.02 | 0.41 | 0.35 | 0.2 | 0.07 | 0.04 | 0.25 |
| a2 | 0.46 | 0.19 | 0.72 | 0.57 | 0.43 | 0.49 | 0.3 | 0.23 | 0.2 | 0.2 | 0.12 | 0.08 | 0.55 | 0.5 | 0.29 | 0.11 | 0.09 | 0.2 |
| a3 | 0.37 | 0.22 | 0.63 | 0.64 | 0.44 | 0.79 | 0.37 | 0.3 | 0.24 | 0.24 | 0.16 | 0.11 | 0.38 | 0.45 | 0.36 | 0.13 | 0.12 | 0.13 |
| a4 | 0.24 | 0.34 | 0.39 | 0.46 | 0.48 | 0.55 | 0.87 | 0.45 | 0.35 | 0.35 | 0.23 | 0.13 | 0.19 | 0.41 | 0.41 | 0.24 | 0.18 | 0.04 |
| a5 | 0.21 | 0.57 | 0.36 | 0.41 | 0.44 | 0.47 | 0.45 | 0.83 | 0.61 | 0.61 | 0.48 | 0.38 | 0.17 | 0.37 | 0.31 | 0.53 | 0.45 | 0.05 |
| a6 | 0.19 | 0.63 | 0.32 | 0.37 | 0.44 | 0.47 | 0.42 | 0.68 | 0.79 | 0.66 | 0.53 | 0.51 | 0.16 | 0.31 | 0.36 | 0.6 | 0.55 | 0.05 |
| a7 | 0.19 | 0.69 | 0.28 | 0.34 | 0.4 | 0.42 | 0.5 | 0.72 | 0.7 | 0.7 | 0.55 | 0.54 | 0.14 | 0.25 | 0.37 | 0.6 | 0.58 | 0.01 |
| e1 | 0.06 | 0.42 | 0.18 | 0.27 | 0.35 | 0.35 | 0.29 | 0.54 | 0.46 | 0.46 | 0.88 | 0.7 | 0.18 | 0.33 | 0.35 | 0.53 | 0.48 | 0.25 |
| e2 | 0.04 | 0.56 | 0.13 | 0.2 | 0.25 | 0.25 | 0.34 | 0.59 | 0.6 | 0.6 | 0.85 | 0.83 | 0.14 | 0.24 | 0.33 | 0.65 | 0.61 | 0.25 |
| g1 | 0.17 | 0.14 | 0.38 | 0.47 | 0.3 | 0.43 | 0.34 | 0.22 | 0.17 | 0.17 | 0.2 | 0.13 | 0.3 | 0.71 | 0.81 | 0.35 | 0.18 | 0.12 |
| s | 0.29 | 0.01 | 0.39 | 0.3 | 0.29 | 0.18 | 0.06 | 0.02 | 0.01 | 0.01 | 0.25 | 0.25 | 0.35 | 0.27 | 0.14 | 0.05 | 0.04 | 0.88 |

Table I
SYNTACTIC SIMILARITY MATRIX FOR BPMN PROCESSES (V1 VS. V2)

All these independent LNT processes are then composed in parallel and synchronized in order to respect the BPMN execution semantics. For instance, after the execution of a node, the corresponding LNT process synchronizes with the process encoding the outgoing flow, which then synchronizes with the process encoding the node appearing at the end of this flow, and so on.

Table II gives examples of encoding patterns in LNT for some BPMN constructs. The actions corresponding to the flows (incf, outf, etc.) will be used as synchronization points between the different BPMN elements. The begin and finish actions in the initial/end events are used to trigger and terminate, respectively, these events. The actions used in task constructs (*e.g.*, task) will be the only ones to appear in the final LTS. All other synchronization actions will be hidden because they do not make sense from an observational point of view. Both the sequence flow and the task construct are enclosed within an LNT loop operator since these elements can be repeated several times if the BPMN process exhibits looping behaviours. The parallel gateway is encoded using the par LNT operator, which corresponds in this case to an interleaving of all flows. The exclusive gateway is encoded using the select LNT operator, which corresponds to a non-deterministic choice among all flows.

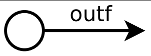
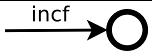
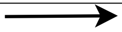
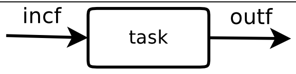
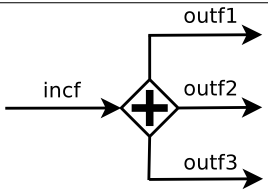
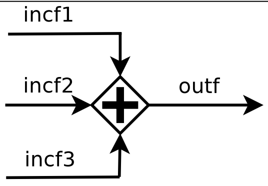
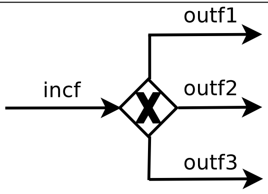
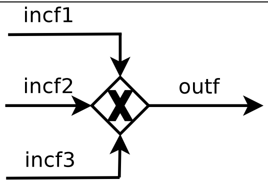
Once all BPMN elements are encoded into LNT, the last step is to compose them in order to obtain the behaviour of the whole BPMN process. To do so, we compose in parallel all the flows with all the other constructs. All flows are interleaved because they do not interact one with another. All events and nodes (start/end events, tasks, gateways) are interleaved too for the same reason. Then both sets are synchronized on flow sequences (incf/outf actions). These additional actions are finally hidden because they should not appear as observable actions and will be transformed into internal transitions in the resulting LTS.

**Example.** Let us first show an excerpt of the LNT encoding for the main process (Figure 3), where we can see that all flows are interleaved as well as all nodes. On the other hand, all flows and nodes synchronize on flow actions. All flows are hidden but task names remain visible and thus appear in the resulting LTS. Figure 4 shows the LTS corresponding to the semantic model of the initial version of the evisa process. This LTS is obtained from the LNT specification by using the CADP compiler. We can see that the LTS exhibits all possible executions of the process. For instance, an exclusive gateway transforms into a state with two outgoing transitions as it is the case for state s3.

## *B. Semantic Measure of Similarity*

Once the processes are translated into LNT and then LTS, the next step is to compare these LTSs. Therefore, in this

Table II
ENCODING PATTERNS IN LNT FOR SOME BPMN CONSTRUCTS

| BPMN construct | BPMN notation | LNT encoding |
|---|---|---|
| Initial event |  | begin ; outf |
| End event |  | incf ; finish |
| Sequence flow |  | **loop** begin ; finish **end loop** |
| Task |  | **loop** incf ; task ; outf **end loop** |
| Parallel gateway (split) |  | incf ; **par** outf1 \|\| outf2 \|\| outf3 **end par** |
| Parallel gateway (merge) |  | **par** incf1 \|\| incf2 \|\| incf3 **end par ;** outf |
| Exclusive gateway (split) |  | incf ; **select** outf1 [] outf2 [] outf3 **end select** |
| Exclusive gateway (merge) |  | **select** incf1 [] incf2 [] incf3 **end select ;** outf |

section, we introduce the measure of similarity between two LTSs [10]. When comparing both LTSs, if the equivalence checker (e.g., Bisimulator [11]) returns *true*, it means that the two LTSs are the same with respect to strong bisimulation [6]. If the equivalence checker indicates that both LTSs are not strongly bisimilar, the similarity measure comes into play to quantify the degree of similarity between the two subparts of both LTSs that are not equivalent (non-bisimilar states). The measure relies on two kinds of criteria, namely global and local criteria, which focus on two non-bisimilar states (one in each LTS).

More precisely, given two LTSs, we first use the partition refinement algorithm [12] to compute bisimilar and non-bisimilar states. Then, we focus on non-bisimilar states and propose a measure comparing all non-bisimilar states according to several local and global criteria. These criteria exploit the different elements present in both LTSs, namely, initial states, labels, structure of the LTS or position of states in the LTS. For each couple of non-bisimular states (one non-bisimilar state from each LTS), we compute a degree of similarity which belongs to [0..1]. All these results are stored in a matrix where non-bisimilar states of one LTS appear in row and non-bisimilar states of the other LTS appear in column.

Given two non-bisimilar states $s_1$, $s_2$ where $s_1 \in LTS_1$ and $s_2 \in LTS_2$, we first compute the similarity of these states using four local criteria. The first one compares outgoing transitions to see the number of matching labels. The second one does the same with incoming transitions. The third one checks whether the nature of states differ (initial or not). The last one compares the similarity of neighbour states.

Global criteria aim at considering the structure of both LTSs and looking at the respective positions of both states in their LTSs. More precisely, there are three global criteria. The first one computes the distance from the initial state to the given state in both LTSs and compares these distances. The second one computes the distance from a given state to the closest bisimilar state and compares these distances. In both cases, we

```
process main [applyOnline:any, uploadPassport:any, ...] is
 hide begin:any, finish:any, flow1_begin:any,
      flow1_finish:any, ... in
    par flow1_begin, flow1_finish, flow2_begin,
         flow2_finish, ... in
     par
      flow[flow1_begin,flow1_finish] ||...||
           flow[flow10_begin,...]
     end par
    ||
    par
        init[begin,flow1_begin] ||
             final[flow6_finish,finish]
     || final[flow10_finish,finish] || ... ||
     || xorsplit_g1[flow4_finish,flow5_begin,flow6_begin]
     || task_a1[flow1_finish,applyOnline,flow2_begin]
     || ...
    end par
   end par
 end hide
end process
```

Figure 3. Main LNT Process for the Evisa Process

compute the shortest distance. The third one aims at comparing whether these paths (from the initial state to the current state or from the current state to the closest bisimilar state) are similar. To do so, we first search for the shortest paths in both LTSs and we compute the number of common labels out of the total number of labels, which gives the main part of this similarity measure. We also add a bonus to this resulting value if the order of labels is the same in both paths.

Given all these values for a couple of states, we can then compute its value in the matrix ($matrix[s_1, s_2]$). This is obtained by using the weighted average of these values (*e.g.*, 1/7 or arbitrary weights). Note that these weights are parameters of the approach. We can decide to change them for putting more emphasis on labels or on some of the local criteria for instance. In the results presented in the paper, we chose equal weights for these parameters.

Since the similarity of neighbour states uses the matrix itself, we use an iterative algorithm that stops when the matrix stabilizes. Once the matrix is computed, we can compute several global measures, which give a degree of semantic similarity of both LTSs. These global measures are computed by using, for instance, the average of the best score for each row and for each column, or the average of scores above a threshold.

**Example.** Figure 5 shows the LTS corresponding to the second version of the evisa process. In the final part of this LTS, we can see how the parallel gateway transforms into an interleaving of all branches involved in this construct.

Table III shows the matrix for the semantic similarity measure for non-bisimilar states only (some states are thus missing such as s4, s7 for V1 and s5, s10 for V2). 79% of states are non-bisimilar and those states are similar to 76% (average of best scores) and to 89% (average of best scores above threshold of 0.8). The matrix shows that some non-bisimilar states are very similar with high values: 0.91 for (s0,s0), 0.85 for (s1,s1) and 0.92 for (s3,s4). The matrix also highlights the main differences between these two models, with lower values for states s2, s3, s6, s7, s8 and s9 (from the V2 LTS point of view).

## C. Mapping

A main problem when reasoning on the semantic model corresponding to the BPMN process is that there is no clear correspondence between the semantic and the syntactic model. This is necessary to make the semantic similarity measure useful in practice to users and to precisely identify the similar or different nodes, from a semantic perspective, in the original BPMN model. Therefore, to that purpose, we present in this section an algorithm that builds a mapping between the LTS states and the BPMN nodes.

Algorithm 1 takes as input a BPMN graph and its corresponding LTS model. As a result, it returns the mapping consisting of a set of couples, where each couple is a correspondence between a node of the BPMN graph and a state of the LTS model. To compute this mapping, the algorithm traverses all nodes (line 2). For each node which is not an activity, it computes all activities that can be executed right after that node (line 4). Then, the algorithm traverses all the states in the LTS (line 5), and for each state whose outgoing labels are the same as the executable activities (line 7), a couple is added to the resulting mapping (line 8). The complexity of this algorithm is $O(N \times S)$ where $N$ is the number of nodes in the BPMN model and $S$ is the number of states in the corresponding LTS model.

**Algorithm 1** ComputeMapping

```
Input: $G = (N, E)$, $LTS = (s^0, S, T, A)$
Output: Mapping $M$ // set of couples (node, state)
 1: $M \leftarrow \{\}$ // initialised to empty set
 2: for each $node$ in $N$ do
 3:   if ($node.getType()$!='Activity') then
 4:     $act \leftarrow node.computeNextActivities()$
 5:     for each $s$ in $S$ do
 6:       $lab \leftarrow s.getOutgoingLabels()$
 7:       if (lab==act) then
 8:         $M.add((node.getIdent(), s.getIdent()))$
 9:         // add a couple to the resulting mapping
10:       end if
11:     end for
12:   end if
13: end for
14: return $M$
```

Note that there is not always a corresponding state for each node in the BPMN graph (activity nodes do not have any counterpart in the LTS for example). Similarly, a state of the LTS model may not have any counterpart in the BPMN graph. Last but not least, this mapping is particularly useful when showing the results of the semantic similarity measure to clearly indicate what node is involved in the process when referring to a state of the LTS model. Typically, when there are two states with a high semantic similarity value for instance,

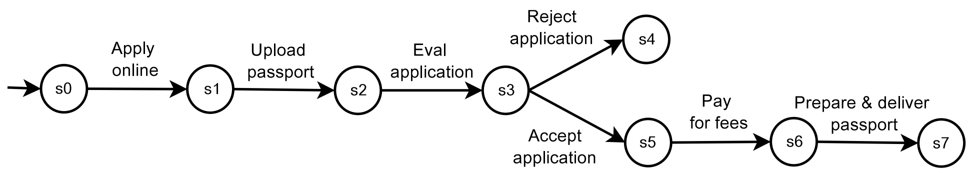


Figure 4. LTS Model of the Evisa Process (V1)

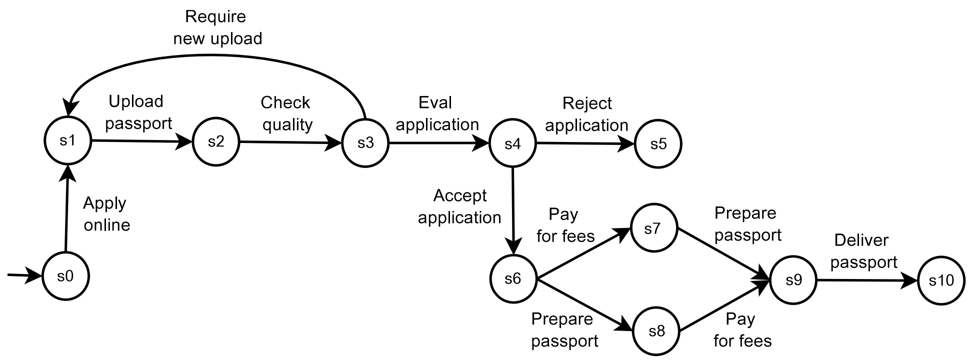


Figure 5. LTS Model of the Evisa Process (V2)

| | s0 | s1 | s2 | s3 | s4 | s6 | s7 | s8 | s9 |
|---|---|---|---|---|---|---|---|---|---|
| s0 | 0.91 | 0.39 | 0.32 | 0.32 | 0.22 | 0.22 | 0.18 | 0.18 | 0.25 |
| s1 | 0.32 | 0.85 | 0.53 | 0.45 | 0.35 | 0.36 | 0.3 | 0.3 | 0.24 |
| s2 | 0.26 | 0.43 | 0.73 | 0.71 | 0.44 | 0.35 | 0.39 | 0.39 | 0.3 |
| s3 | 0.19 | 0.35 | 0.42 | 0.52 | 0.92 | 0.35 | 0.34 | 0.34 | 0.4 |
| s5 | 0.17 | 0.3 | 0.36 | 0.45 | 0.42 | 0.75 | 0.45 | 0.66 | 0.35 |
| s6 | 0.24 | 0.25 | 0.29 | 0.35 | 0.47 | 0.4 | 0.58 | 0.44 | 0.58 |

Table III
SEMANTIC SIMILARITY MATRIX FOR BPMN PROCESSES (V1 VS. V2)

the mapping allows us to show the corresponding nodes in both processes.

**Example.** We illustrate the mapping computation with the second version of the evisa process. Figure 6 shows the relationship computed by our algorithm, which helps the user relate the states in the semantic model to the nodes in the BPMN process. Interestingly, we see that all start/end nodes and split/join gateways in the BPMN process have a counterpart in the LTS. Activities do not have any corresponding states because labels are not associated to states but to transitions in the corresponding LTS. There are also states without any corresponding nodes. This is the case for example for state s2, which corresponds to the sequence of two activities. Suppose now that the user is interested in checking the nodes corresponding to states (s3, s4) whose semantic similarity value is 0.92 in Table III. The mapping says that s3 corresponds to node g1 in the first version of the evisa process whereas s4 corresponds to node g3 in the second version of the evisa process. By looking at these two nodes, the user can easily understand this high value computed by the semantic similarity measure.

## V. TOOL SUPPORT

In this section, we present the tools supporting the different steps of our approach. Note that the whole process is entirely automated (no human intervention required) as shown in Figure 7. Computation of syntactic similarity is achieved with a new tool we have implemented in Python using the results presented in Section III. Computation of semantic similarity relies on the combination of three tools, two existing ones (VBPMN [2] and DLTS [10]) and a new one for computing the mapping. VBPMN allows the transformation from BPMN processes to LTS models via a process algebra encoding. DLTS takes LTS models as input and return a similarity measure between these two LTSs. The mapping construction has been implemented in Python as a new tool. It takes one BPMN process and its corresponding LTS model as input, and returns a correspondence between the syntactic and the semantic models as presented in Section IV.

**Experimental results.** We applied the syntactic and semantic similarity tools to several examples. Each example consists of two versions of a BPMN process, which present some differences in terms of tasks and structures. These experiments

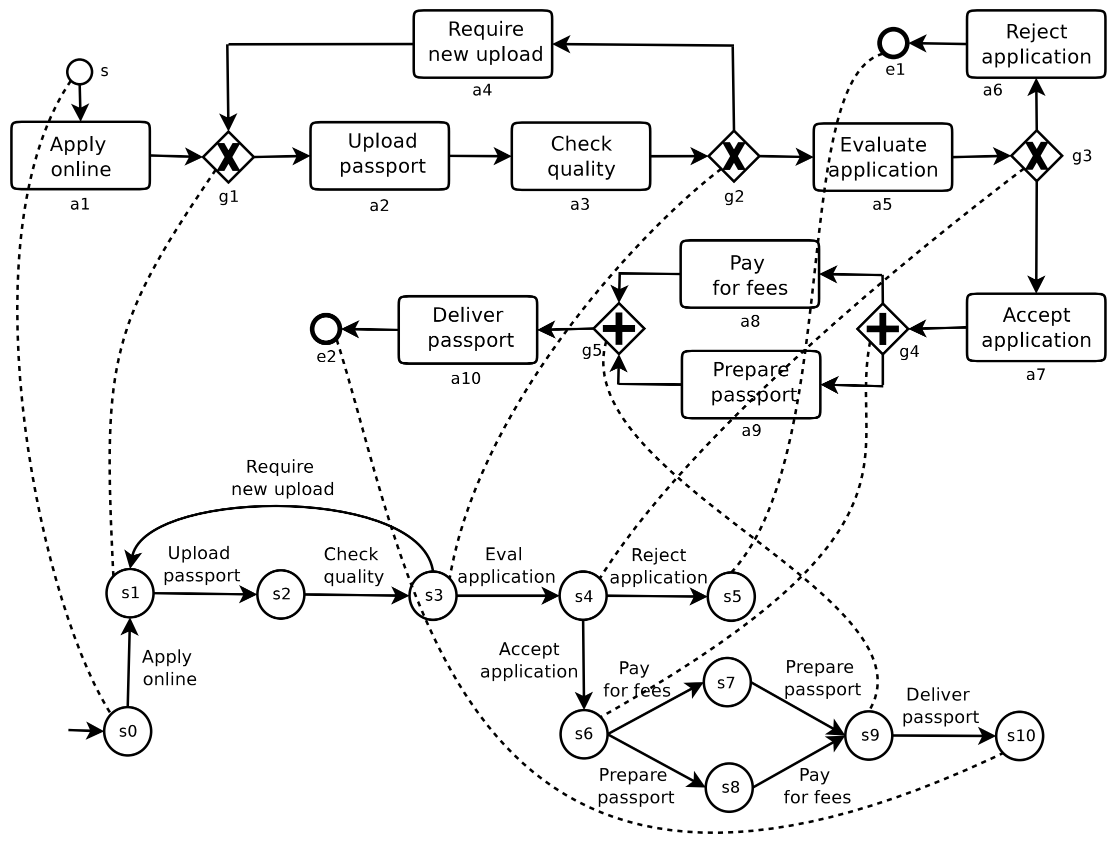


Figure 6. Mapping between BPMN Process and LTS Model for the Evisa Service (V2)

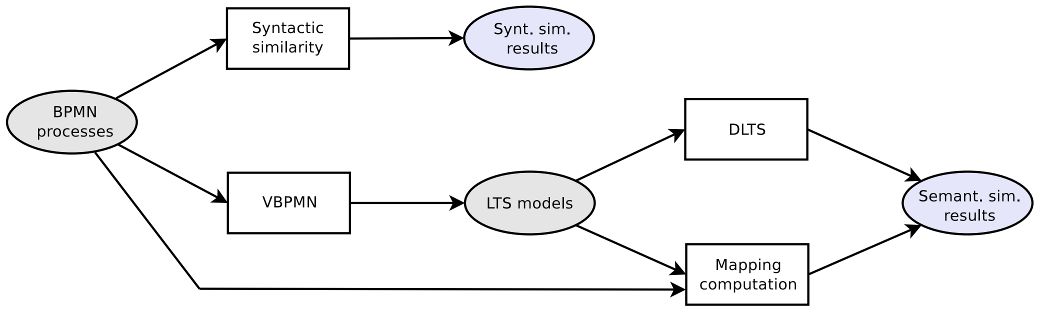


Figure 7. Tool Support Overview

have been carried out to evaluate the quality of the approach in terms of performance (computation time) and accuracy of the results. Table IV shows some of these experiments. The first example is the running example used in this paper. The second example is a travel agency taken from [13]. The third example is an employee hiring process presented in [14]. The fourth example is an account opening process and the fifth an incident management system, both taken from the VBPMN suite [2], [15]. The second and third columns of Table IV characterize the example in terms of number of nodes/flows for the process and number of states/transitions for the corresponding LTS model. The three following columns describe the resulting syntactic and semantic global measures, the times it takes to compute the similarity results, and the accuracy of the results expressed using the recall and precision measures.

First of all, one can see that all these examples are rather similar syntactically and semantically speaking (high values in the global measures column), which is the case in practice since each example consists of two versions of a same case study. Using these techniques and tool on completely different BPMN processes would not be interesting or useful, because it would return very low values of similarity.

Let us now comment on performance. It takes milliseconds to compute both similarity measures for the examples in the table. Note that VBPMN takes a few seconds to generate the LTS corresponding to the BPMN process (see [2] for detailed

Table IV
EXPERIMENTAL RESULTS.

| Ex. | Process |  | LTS |  | Global Meas. |  | Time |  | Accuracy |  |
|---|---|---|---|---|---|---|---|---|---|---|
|  | Nodes | Flows | States | Trans. | Synt. | Sem. | Synt. | Sem. | Synt. | Sem. |
| P1 | 11 | 10 | 8 | 7 | 85% | 89% | 0.24ms | 0.18ms | 82% | 88% |
| P1' | 18 | 19 | 11 | 12 |  |  |  |  |  |  |
| P2 | 10 | 11 | 6 | 7 | 95% | 93% | 0.13ms | 0.15ms | 90% | 83% |
| P2' | 13 | 15 | 9 | 12 |  |  |  |  |  |  |
| P3 | 20 | 23 | 10 | 15 | 91% | 93% | 0.8ms | 0.3ms | 95% | 90% |
| P3' | 22 | 26 | 14 | 24 |  |  |  |  |  |  |
| P4 | 25 | 29 | 20 | 32 | 95% | 92% | 1.6ms | 0.8ms | 100% | 100% |
| P4' | 28 | 33 | 20 | 33 |  |  |  |  |  |  |
| P5 | 14 | 16 | 7 | 8 | 88% | 97% | 0.3ms | 0.1ms | 85% | 100% |
| P5' | 18 | 22 | 8 | 11 |  |  |  |  |  |  |

experiments on this part of the approach), and this time is not included in the table. The examples presented in the table are of medium-size (less than one hundred of nodes and flows). If one wants to analyze very large processes consisting of hundreds or thousands of flows/nodes, the computation of both similarity measures can take several seconds for such processes. This is due to the DLTS tool, which takes time to compute the resulting matrix for large LTSs. Note that the semantic model (LTS) can be much larger than the original BPMN process.

We finally focus on the evaluation of the accuracy of the results computed by our approach. To do so, we rely on the well-known precision and recall measures [16]. Comparison of process models involves matching, that is, the detection of correspondences between nodes (in the case of syntactic comparison) or states (in the case of semantic comparison), which is considered as basis here for computing the precision and recall measures. A syntactic (semantic, resp.) match between nodes (states, resp.) is returned for the highest value in the matrix from the point of view of the smaller process (LTS, resp.). Recall is the number of correct identified matches out of the total number of matches returned by the approach. Precision is the number of correct identified matches out of the total number of identified matches. In our case, the total number of matches returned by the approach and the total number of identified matches coincide because we return one match per node/state for the smaller process/LTS. Therefore, we give in Table IV a single value of accuracy for the syntactic and semantic measures. As one can see in the table, the accuracy values are very high (between 80 and 100%) for both measures.

## VI. RELATED WORK

In this section, we focus on existing approaches for comparing BPMN processes (or workflows). In [17], the authors propose a theoretical framework for comparing BPMN processes. Their main focus is substitutability and therefore they explore various sorts of behavioural equivalences in order to replace equals for equals. This work applies at the BPMN level and aims at detecting equivalent patterns in processes. In a related line of works, [18] studies BPMN behaviours from a semantic point of view. It relies on notions of equivalence as those introduced in [17]. It presents several BPMN patterns and structures that are syntactically different but semantically equivalent. This work is not theoretically grounded and is not complete in the sense that only a few patterns are tackled. The authors of [18] also overview best practices that can be used as guidelines by modelers for avoiding syntactic discrepancies in equivalent process models. Compared to our approach, this work relies on strong notions of equivalence and the behaviour is preserved only if such an equivalence is verified. We prefer in this work to quantify the differences to provide a finer measure of the discrepancies between two process models.

BPMNDiffViz [19] was developed as a tool, which helps users to find structural differences between two business processes represented in BPMN format. The differences are visualized and the user can navigate between them. BPMN-DiffViz shows, which elements match, should be added, or should be deleted to transform one process model to another. Here we go beyond syntactic comparison by providing a semantic similarity measure too. In [20], the authors address the equivalence or alignment of two process models. To do so, they check whether correspondences exist between a set of activities in one model and a set of activities in the other model. They consider Petri nets as input and process graphs as low-level formalism for analysis purposes. Their approach resides in the identification of regions (set of activities) in each graph that can coincide with respect to an equivalence notion. They particularly study two equivalence notions, namely trace and branching equivalences. This approach does not work in the presence of overlapping correspondences, meaning that in some cases, the input models cannot be analyzed. This work shares similarities with our approach, in particular the use of low-level graph models, hiding techniques and behavioural equivalences for comparing models. In addition, our approach provides quantitative results for supporting the comparison process.

Several works aim at measuring the degree of similarity of business process models, see, *e.g.*, [21]–[23]. [21] uses causal footprints as an abstract representation of the behaviour captured by a process model. Then, given two footprints, the similarity is computed using the vector space model approach from information filtering and retrieval. [22] relies on Petri nets and propose to compare process models by analysing event logs with typical execution sequences. This paper presents several notions of comparison, such as precision and recall. Precision measures indicate whether the second model's behaviour is possible according to the first model's behaviour. Recall measures are used to quantify how much of the first model's behaviour is covered by the second model. [23] tackles the problem of retrieving process models from a repository that most closely resemble a given process model. To do so, the authors present three similarity metrics that can be used to answer these queries: (i) label matching similarity that compares the labels attached to process model elements; (ii) structural similarity that compares element labels as well as the topology of process models using graph edit distance; and (iii) behavioural similarity that compares element labels as well as causal relations captured in the process model using causal footprints as in [21]. Compared to these works, the main differences is that we provide a different solution for behavioural similarity, which relies on LTS semantics for BPMN and LTS comparison. Moreover, we provide a relationship between the syntactic and the semantic model thus allowing users to relate the differences and similarities in the semantic model with the original BPMN process.

## VII. Concluding Remarks

In this paper, we have presented new techniques for measuring the similarity of BPMN processes from a syntactic and semantic point of view. Syntactic similarity takes the structure of processes into account and relies on several local and global criteria. Semantic similarity considers an LTS-based semantics for BPMN processes, and relies on LTS comparison for quantifying the difference between two BPMN processes. We have also defined an algorithm for relating the semantic LTS model with the BPMN process given as input. This allows one to relate the results given by the semantic similarity measure with the syntactic model. The whole approach is supported by a set of tools, some already existing and some we implemented. We carried out several experiments to show that performance and accuracy of the computed results are satisfactory.

As far as future work is concerned, we plan to extend this work to more expressive BPMN models, taking into account data-based conditions or probabilities of execution for exclusive gateways, see [24]–[26] for instance. This would require to entirely revise our approach for computing similarity. Another idea is to take advantage of the similarity measures for supporting the evolution of a process by guaranteeing that the new version of the process extends the original one with respect to some defined evolution properties. A last perspective is to better take advantage of all similarity values computed by our approach and provide user-friendly visualization techniques for these results.

## References

[1] ISO/IEC, "International Standard 19510, Information technology – Business Process Model and Notation," 2013.

[2] A. Krishna, P. Poizat, and G. Salaün, "Checking business process evolution," *Sci. Comput. Program.*, vol. 170, pp. 1–26, 2019.

[3] M. Weidlich, J. Mendling, and M. Weske, "A foundational approach for managing process variability," in *Proc. of CAiSE 2011*, ser. LNCS, vol. 6741. Springer, 2011, pp. 267–282.

[4] M. Kunze, M. Weidlich, and M. Weske, "Behavioral similarity - A proper metric," in *Proc. of BPM'11*, ser. LNCS, vol. 6896. Springer, 2011, pp. 166–181.

[5] M. Ouederni, U. Fahrenberg, A. Legay, and G. Salaün, "Compatibility Flooding: Measuring Interaction of Services Interfaces," in *Proc. of SAC'17*. ACM, 2017, pp. 1334–1340.

[6] R. Milner, *Communication and concurrency*. Prentice Hall, 1989.

[7] D. Champelovier, X. Clerc, H. Garavel *et al.*, "Reference Manual of the LNT to LOTOS Translator (Version 6.7)," 2018, 153 pages.

[8] ISO, "LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour," ISO, Tech. Rep. 8807, 1989.

[9] H. Garavel, F. Lang, R. Mateescu *et al.*, "CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes," *STTT*, vol. 15, no. 2, pp. 89–107, 2013.

[10] G. Salaün, "Quantifying the similarity of non-bisimilar labelled transition systems," *Sci. Comput. Program.*, vol. 202, 2021.

[11] D. Bergamini, N. Descoubes, C. Joubert, and R. Mateescu, "Bisimulator: A Modular Tool for On-the-Fly Equivalence Checking," in *Proc. of TACAS'05*, ser. LNCS, vol. 3440. Springer, 2005, pp. 581–585.

[12] R. Paige and R. E. Tarjan, "Three Partition Refinement Algorithms," *SIAM J. Comput.*, vol. 16, no. 6, pp. 973–989, 1987.

[13] P. Poizat and G. Salaün, "Checking the Realizability of BPMN 2.0 Choreographies," in *Proc. of SAC'12*. ACM Press, 2012, pp. 1927–1934.

[14] F. Durán and G. Salaün, "Verifying Timed BPMN Processes Using Maude," in *Proc. of COORDINATION'17*, ser. LNCS, vol. 10319. Springer, 2017, pp. 219–236.

[15] A. Krishna, P. Poizat, and G. Salaün, "VBPMN: Automated Verification of BPMN Processes," in *Proc. of IMF'17*, ser. LNCS, vol. 10510. Springer, 2017, pp. 323–331.

[16] K. M. Ting, "Precision and Recall," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Springer US, 2010, pp. 781–781.

[17] V. Lam, "Foundation for Equivalences of BPMN Models," *Theoretical and Applied Informatics*, vol. 24, no. 1, pp. 33–66, 2012.

[18] K. Kluza and K. Kaczor, "Overview of BPMN Model Equivalences. Towards Normalization of BPMN Diagrams," in *Proc. of KESE'12*, 2012, pp. 38–45.

[19] S. Ivanov, A. A. Kalenkova, and W. M. P. van der Aalst, "BPMNDiffViz: A Tool for BPMN Models Comparison," in *Proc. of BPM'15 (Demo Session)*, ser. CEUR Workshop Proceedings, F. Daniel and S. Zugal, Eds., vol. 1418, 2015, pp. 35–39.

[20] M. Weidlich, R. M. Dijkman, and M. Weske, "Behaviour Equivalence and Compatibility of Business Process Models with Complex Correspondences," *Comput. J.*, vol. 55, no. 11, pp. 1398–1418, 2012.

[21] B. F. van Dongen, R. M. Dijkman, and J. Mendling, "Measuring Similarity between Business Process Models," in *Proc. of CAISE'08*, 2008, pp. 450–464.

[22] A. K. A. de Medeiros, W. M. P. van der Aalst, and A. J. M. M. Weijters, "Quantifying Process Equivalence Based on Observed Behavior," *Data Knowl. Eng.*, vol. 64, no. 1, pp. 55–74, 2008.

[23] R. M. Dijkman, M. Dumas, B. F. van Dongen, R. Käärik, and J. Mendling, "Similarity of business process models: Metrics and evaluation," *Inf. Syst.*, vol. 36, no. 2, pp. 498–516, 2011.

[24] 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*, ser. LNCS, vol. 11152. Springer, 2018, pp. 76–97.

[25] ——, "A Rewriting Logic Approach to Resource Allocation Analysis in Business Process Models," *Sci. Comput. Program.*, vol. 183, 2019.

[26] Y. Falcone, G. Salaün, and A. Zuo, "Probabilistic Model Checking of BPMN Processes at Runtime," in *Proc. of IFM'22*, ser. LNCS, vol. 13274. Springer, 2022, pp. 191–208.