"Detecting Frequently Recurring Structures in BPMN 2.0 Process Models"

Marigianna Skouradaki, Frank Leymann

Institute of Architecture of Application Systems,
University of Stuttgart, Germany
firstname.lastname@iaas.uni-stuttgart.de

```
@inproceedings{skourama_summersoc_2015,
  author    = {Marigianna Skouradaki and Frank Leymann},
  title     = {Detecting Frequently Recurring Structures in BPMN 2.0
               Process Models},
  booktitle = {Proceedings of the 9th Symposium and Summer School On
               Service-Oriented Computing: SummerSOC'15},
  editors   = {Christos Nikolaou and Frank Leymann},
  year      = {2015},
  pages     = {102--116},
  publisher = {IBM}
}
```

Detecting Frequently Recurring Structures in BPMN 2.0 Process Models

Marigianna Skouradaki[1] and Frank Leymann[1]

Institute of Architecture of Application Systems, University of Stuttgart, Germany
{skouradaki,leymann}@iaas.uni-stuttgart.de

Abstract Reusability of process models is frequently discussed in the literature. Practices of reusability are expected to increase the performance of the designers, because they do not need to start everything from scratch, and the usage of best practices is reinforced. However, the detection of reusable parts and best practices in collections of BPMN 2.0 process models is currently only defined through the experience of experts in this field. In this work we extend an algorithm that detects the recurring structures in a collection of process models. The extended algorithm counts the number of times that a recurring structure appears in a collection of process models, and assigns the corresponding number to its semantics. Moreover, the dublicate entries are eliminated from the collection that contains the extracted recurring structures. In this way, we assert that the resulting collection contains only unique entries. We validate our methodology by applying it on a collection of BPMN 2.0 process models and analyze the results. As shown in the analysis the methodology does not only detect applied practices, but also leads to conclusions of our collection's special characteristics.

Keywords: BPMN 2.0, Relevant Process Fragments, RPF, process models, reusabulity, structural, similarity

1 Introduction

During the last years many researchers [6, 7, 12, 15, 16] have emphasized the importance of the reusability of process models. It is expected that an efficient methodology to reuse process models will contribute to a more effective engineering of process models [15]. For this reason we need to analyze the process models

and decide which parts can be reused. The research field of process models similarities focuses on three different areas: 1) text semantics, 2) structural analysis and 3) behavioral analysis [5].

Large collections of process models are anonymized or modeled for documentation. Thus, they are not executable. Consequently, the approaches of text semantics (vs. anonymized models) and behavioral analysis on executable models (vs. mock-up, non-executable models) cannot be used efficiently. In these cases we need to apply the approach of structural similarities. However, structural similarities also base their functionality on text semantics and behavioral similarities [4, 5, 14]. For this reason we have suggested a methodology that runs (sub)graph isomorphism against a collection of process models and focuses on extracting the common recurring structures. The first approach of our methodology leads to promising results, as experiments showed that the algorithm can run with logarithmic complexity [17].

This work extends the work described in [17] with the following contributions:

1. extending the algorithm to count the recurring structures and filter the duplicate results
2. applying the algorithm on different use case scenarios
3. analyzing the exported results

This paper is structured as follows: section 2 describes the overall methodology overview of our work. Section 3 defines the problem and explains the basic concepts that frame it. Section 4 shows the implementation of the designed methodology. Section 5 discusses the results of the methodology's application. Section 6 addresses the related work in this area, and Section 7 concludes and describes our plans for future work.

2 Methodology Overview

The BenchFlow Project[1] aims to create the first benchmark for Business Process Model and Notation (BPMN 2.0) compliant Workflow Engines. In the scope of that project we have collected process models that reflect the diversity of application scenarios. For the construction of a representative benchmark it is needed to extract the essence of each of the scenarios and construct a set of representative process models. The extracted representative process models

[1] http://www.iaas.uni-stuttgart.de/forschung/projects/benchflow.php

are called “synthetic” process models, because even though they are artificial they constitute an accurate representation of real world use cases. Synthetic process models should also be combined with appropriate Web Services, synthetic data and interacting users. To address these challenges, we develop a workload generator. In this work we focus on the methodology to synthesize representative process models while the generation of appropriate web services and interacting users is left for future work.

Figure 1 depicts the methodology for the generation of the synthetic processes, which uses the following four phases:

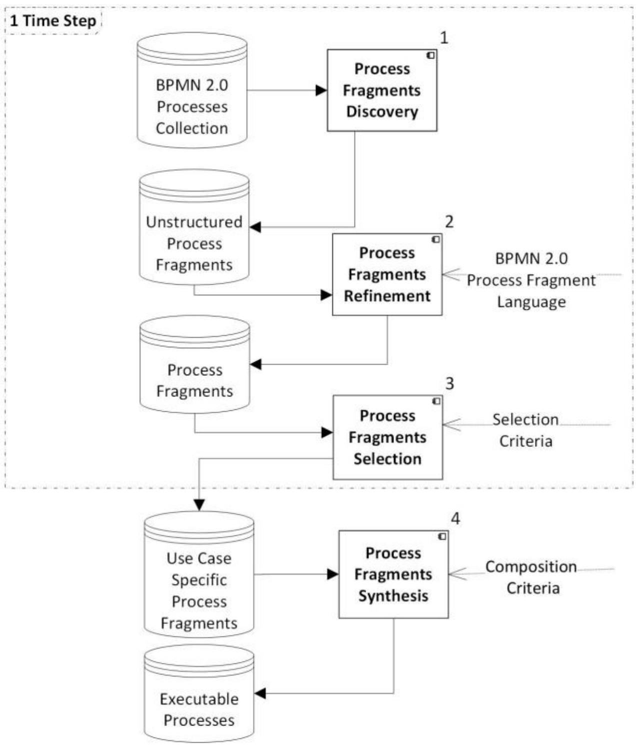
1 Time Step
BPMN 2.0 Processes Collection
1
Process Fragments Discovery
Unstructured Process Fragments
2
Process Fragments Refinement
BPMN 2.0 Process Fragment Language
Process Fragments
3
Process Fragments Selection
Selection Criteria
Use Case Specific Process Fragments
4
Process Fragments Synthesis
Composition Criteria
Executable Processes

Figure 1: Methodology Overview

1. **Process Fragments Discovery**: Addresses the automatic discovery of recurring structures in a collection of process models. Our methodology is applied in a collection of BPMN 2.0 process models. The definition and implementation of this methodology are described in [17].

2. **Process Fragments Refinement**: The extracted recurring structures are stored as unstructured BPMN 2.0 code. They need to be refined as "Process Fragments" [15] in order to be stored in a process fragment library, out of which they can be managed, and retrieved.
3. **Process Fragments Selection**: All process fragments are not necessarily of equal interest for the benchmark. For example, in a benchmark scenario we are interested in high parallel fragments, while in another scenario we need to check how the Workflow Engine responds to complex control flow branching structures. This component selects fragments that satisfy benchmark related criteria. The criteria are defined by the user or from the benchmark customization. The work described in this paper focuses on this component, because it calculates the appearance rate of each recurring structure in the collection. Furthermore, it annotates the recurring structure with its appearance number. This is one of the metrics that will be used for the selection of the process fragments. Other metrics that we defined are size, structural metrics, metrics of external interaction, data handling and complexity [2,13]. The process fragments that we select from this component are stored in a separate repository.

   It is possible that phases 1-3 (Figure 1) can sometimes be omitted if the extraction criteria are compliant with the purposes of the benchmarking process.
4. **Process Fragments Synthesis**: Synthesizes the process fragments into executable processes according to the composition criteria given by the user or the benchmark customization. For example, when the selection criteria ask for a process with control-flow nesting $\leq$ N and M external interactions, the appropriate fragments are chosen to synthesize it.

3 Background

3.1 Problem Definition

In graph theory the task of discovering similar structures is expressed as sub-graph discovery problem. Recurring sub-graph discovery is a sub-category of the general problem of subgraph isomorphism which is proven to be NP-Complete [1, 9]. However there are special cases of graphs and matching problems that are proven to be of lower complexity [20].

Figure 2 presents two process models expressed in BPMN 2.0. As seen, these models consist of nodes (in BPMN 2.0 tasks, events and gateways), directed edges (in BPMN 2.0 sequence flows), and labeling (BPMN 2.0 language semantics on events and gateways, and names on tasks). Hence, process models are a special type of directed attributed graphs. These are graphs where their vertexes or edges contain information.

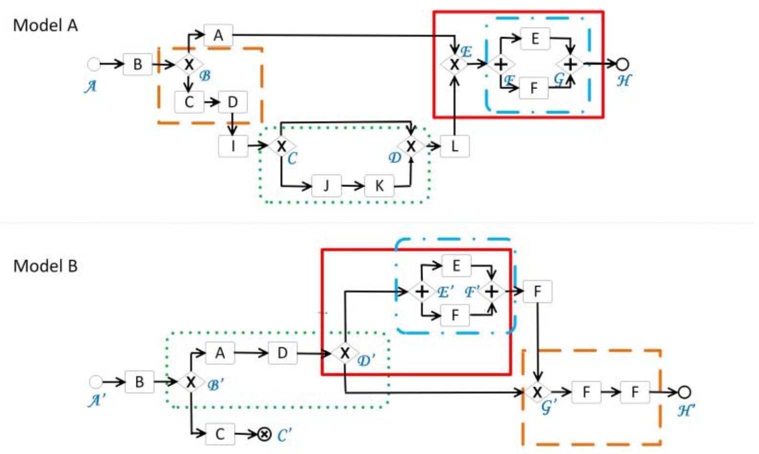

Figure 2: Recurring Structures in two BPMN 2.0 Process Models

Some results of the recurring structures extraction is shown in Figure 2. It shows four pairs of recurring structures in two BPMN 2.0 process models. These structures are not the complete set of reoccurring structures in these two process models. However we considered these as representative cases as they demonstrate the following attributes:

1. Structures can be nested within each other. This means that a recurring structure may be a subgraph of another, bigger recurring structure. Example structures of this attribute are marked with the red solid and the blue dash-dotted line.
2. Reoccurring structures can appear in different positions of the process model. As position we define the number of edges from the node to the model's start event This attribute applies to all represented structures.
3. Structures can be "partially" similar. This means that some of the outgoing edges of a node can lead to similar structures. Structures marked with the orange dash line and the green dotted line are some examples of this attribute.

It is also possible that a structure demonstrates more than one of the above attributes. For example the structure marked with the blue dash-dotted line is nested (Attribute 2) and appears in different positions of the diagram (Attribute 3).

We have used these attributes as a basis to develop the methodology to detect and extract recurring structures [17]. The goal of this work is to: a) detect the duplicate structures and b) to tag them with the appropriate number of appearances. This task is also reduced to a subgraph isomorphism challenge, since we have to reapply the methodology of graph isomorphism in order to decide if a structure is duplicate in the collection.

3.2 Basic Concepts

This section sets the theoretical work used by our methodology of fragmentation, duplicate filtering, and appearance counting. We extend the definition of "Process Fragment", initially given by [15] to fit our needs.

Before we proceed to the extension of the "Process Fragment" definition, we need to define the concept of a "Checkpoint". As "Checkpoint" we define any type of node that can be used as a starting point for our extended process fragments. The types of checkpoints can be configured by the user, and vary with respect to the process language that describes the models.

The extended definition of "Process Fragment" is called a "Relevant Process Fragment" (RPF) because its detection is dependent to its existence in at least K process models. RPF satisfy the following structural requirements:

- starts with a checkpoint
- has at least N nodes including the checkpoint, where N is a natural number and pre-configured from the user. We use $N \geq 3$.
- contains at least one activity

For this paper we focus on the detection of RPFs when $K = 2$, while in future work the threshold of K may vary.

Finally, we define as duplicate any structure that appears in $K + 1$ process models. Since in this work the threshold is set as $K = 2$, a duplicate must appear in at least 3 models before it is filtered.

4 Implementation

4.1 Algorithm: Get Recurring Structures

This section describes the methodology we developed for the discovery of the RPFs. For a better comprehension the algorithm is separated in two algorithms. Algorithm 1 starts the traversal of the model, calls Algorithm 2, and returns the discovered RPF. Algorithm 2 compares two models with each other and extracts the discovered similar structures.

Algorithm 1 takes as input one set of sequence flows (graph's edges) for each model. These sequence flows correspond to every outgoing sequence flows of each checkpoint. Figure 2 shows the scenario were the configured types of checkpoints are gateways and events. For example, see checkpoint $\mathcal{B}$ and checkpoint $\mathcal{E}'$ (cf. figure 2) . The corresponding sets given as input to the algorithm will be $Set1 = \{\{exclusive\ gateway \rightarrow Task\ A\ \}, \{exclusive\ gateway \rightarrow Task\ C\ \}\}$ for Model A and $Set2 = \{\{parallel\ gateway \rightarrow Task\ E\ \}, \{parallel\ gateway \rightarrow Task\ F\ \}\}$ for Model B. These sets of checkpoint sequence flows are called checkpointFlows in lines 3-4 of algorithm 2. This process is done to ensure that all possible sets of paths will be traversed by the end of the algorithm.

Algorithm 1 Procedure that calculates the matching paths of two models

```
1:  procedure getMatchingPath()
2:      $i \leftarrow 0$
3:      for all $checkpointFlows\ chFlowA \in modelA$ do
4:          for all $checkpointsFlows\ chFlowB \in modelB$ do
5:              $tmpFragment \leftarrow \emptyset$
6:              $tmpFragment \leftarrow$ comparison($chFlowA, chFlowB, tmpFragment$)
7:              if isValidFragment($tmpFragment$) then
8:                  add($collection,\ tmpFragment$)
9:              end if
10:         end for
11:     end for
12: end procedure
```

For all possible pairs of checkpointFlows in the two checkpoint sets, namely for their Cartesian product, we call algorithm 2 (line 6 of algorithm 1). It takes as parameters the current instances of checkpointFlows, and an empty fragment. By iteratively calling algorithm 2 for the different checkpointFlows we manage to check all possible checkpointFlows combinations as a possible start point of an RPF. The comparison procedure returns the calculated RPF (cf. variable tmpFragment). At this point we need to validate if the returned structure comprises an RPF or not (cv. function ISVALIDFRAGMENT(tmpFragment)). The validation rules applied are the requirements described in Section 3.2. If the validation is successful, then the RPF is saved in a temporary data structure that holds all the discovered RPFs (cf. variable “collection” in algorithm 1)).

Algorithm 2 Procedure that compares the two models

```
 1: procedure comparison($sequenceFlowModelA, sequenceFlowModelB,$
    $tmpFragment$)
 2:     if (getSourceType($sequenceFlowModelA$)
            equals
            getSourceType($sequenceFlowModelB$))
            and
            (getTargetType($sequenceFlowModelA$)
            equals
            getTargetType($sequenceFlowModelB$)) then
 3:         $outgoingA \leftarrow$ getOutgoing(sequenceFlowModelA)
 4:         $outgoingB \leftarrow$ getOutgoing(sequenceFlowModelB)
 5:         add($fragment, sequenceFlowModelB$)
 6:         for all $outgoingA \in ModelA$ do
 7:             for all $outgoingB \in ModelB$ do
 8:                 if $outgoingB \notin tmpFragment$ then
 9:                     comparison(outgoingA, outgoingB, tmpFragment)
10:                 end if
11:             end for
12:         end for
13:     else
14:         return $fragment$
15:     end if
16: end procedure
```

The functionality of algorithm 2 is to traverse and compare the models. The first step in this procedure is to check if the sequence flows that are given as parameters have sources and targets that are of the same type (e.g. task, exclusive gateway, start event etc.)(cf. functions GETSOURCETYPE() and GETTARGETTYPE()). When this condition is satisfied we say that two sequence flows match. Since we are strictly focusing on the structural similarity of the models this decision can be taken only based on the type of the sequence flow’s source and target node. However, the condition could be extended if we wanted to check the similarity regarding more parameters e.g. labels of the nodes. When the sequence flows match we add the respective sequence flow to a temporary fragment structure (cf. variable tmpFragment in algorithm 2). Afterwards we get all the outgoing sequence flows (cf. function GETOUTGOING()) of the previously considered target nodes, i.e. make a step forward to the traversal, and call recursively the comparison algorithm for all pairs of outgoing sequence flows. The termination condition of the recursion is two sequence flows that do not match, or if the checked nodes do not have any outgoing sequence flows. If this condition is satisfied the algorithm returns the set of matched sequence flows until this point of execution (cf. variable tmpFragment in algorithm 2).

4.2 Algorithm: Find Duplicates and Count Appearance

This section presents the extensions that we developed for the algorithms of subsection 4.1. Their goal is to filter the duplicate RPFs, and count the times of their appearance. Algorithm 3 will check a set of newly discovered RPFs against the collection of stored RPFs, to find the number of appearances of an RPF in the collection. To this end, it calls algorithm 4 for their Cartesian product to check if they are isomorphic. Algorithm 4 checks if two RPFs are isomorphic to each other.

Algorithm 3 Procedure that checks if a fragments is duplicate and increases appearance counter

```
procedure handleDuplicates($matches$)
    for all $match \in matches$ do
        if $collection$ equals $\emptyset$ then
            add(collection,match)
        else if contains($collection, match$) then
            $tmpFragment \leftarrow collection[match]$
            $appearanceCounter \leftarrow tmpFragment.appearanceCounter$
            $tmpFragment.appearanceCounter \leftarrow appearanceCounter + 1$
            $tmpFragment.isomorphic \leftarrow true$
        else
            add(collection,match)
        end if
    end for
end procedure
```

More particularly, algorithm 3 takes as parameter a set of matched RPFs. This is basically the output of algorithm 1 described in subsection 4.1. As discussed the set of newly matched RPFs (cf. *matches*) is compared against the RPFs that are already stored in the *collection*. Firstly, we need to check if the *collection* is empty (cf. variable *collection*). In this case we add the first matched RPF the *collection* and we do not need apply further checks. For every other newly matched RPF (cf. variable *match*) we check if it is contained in the *collection*. The function *CONTAINS* (cf. line 5 in algorithm 3) will internally call algorithm 4 to compare each matched RPF *match* with each RPF stored in the *collection*. If we find an isomorphic match we need to edit its corresponding isomorphic RPF that is stored in the *collection*. We mark it as isomorphic, and we increase its appearance counter by 1. If we do not find any isomorphic match then the RPF is added in the *collection* (cf. function ADD at line 11 of algorithm 3). If we apply this procedure before storing each RPF, the final *collection* will not contain any duplicates.

Algorithm 4 Procedure that calculates if two RPFs are equal

```
1: procedure isFragmentIsomorphic($fragment1, fragment2$)
2:   $fragment1SequenceFlows \leftarrow sequenceFlows \in fragment1$
3:   $fragment2SequenceFlows \leftarrow sequenceFlows \in fragment2$
4:   if $fragment1SequenceFlows.size()$ equals
         $fragment2SequenceFlows.size()$ then
5:       $tmpFragment \leftarrow \emptyset$
6:       comparison($fragment1SequenceFlows[0]$,
          $fragment2SequenceFlows[0]$, $tmpFragment$)
7:       if $tmpFragment.size$ equals $fragment2SequenceFlows.size$ then
8:           $return\ true$
9:       end if
10:  end if
11:  $return\ false$
12: end procedure
```

Algorithm 4 is called internally from the *CONTAINS* function of algorithm 3. Its parameters are the RPFs to compare. It will return true if and only if the two RPFs are completely isomorphic. So the cases of isomorphic subsets are not considered in this case. We first check if the two RPFs have the same number of sequence flows. In this case we call the algorithm 2 which operates as it was described in subsection 4.1. Then we need to check if the output of algorithm 2 has the same size of sequence flows. If two RPFs are isomorphic and have the same number of sequence flows then it is sure that the RPFs are equal. If the two RPFs did not have the same number of sequence flows we know for sure they were not completely isomorphic. In this case false is returned.

5 Validation and Discussion

This section shows and analyzes the results of the methodology's application. For the validation we used 43 BPMN 2.0 process models that originate from the sets of BPMN 2.0 process models[2] also used in [14], and the BPMN 2.0 standard examples[3] [11]. This is a combination of artificial and real-world process models. The RPFs discovery algorithm (cf. algorithm 1) is configured to calculate the types of $Set1 = \{events, gateways\}$ as checkpoints, and we set $N = 3$ the number of nodes of each RPF.

Each model is compared with all the other models except for themselves. This leads to 903 comparisons. That results in 1544 non-filtered RPFs. The results are decreased to 259 RPFs after the filtering of duplicates. This leads to 83.22% decrease of the results. This is an important percentage of decrease because it will help us reduce the exported results and ease the analysis of the produced RPF collection.

The calculated times of appearance of the detected RPFs range from $[1, 178]$. From the RPFs that appear more than one times in the collection (54 RPFs) we calculate the median value and set it as threshold. Statistically the median value shows a central tendency of a set. Hence, it was considered representative as a threshold. In this case:

$$Median = Threshold = 14$$

We result in 27 RPFs with re-appearance rate above the $threshold$. Setting the $threshold$ does not only eliminate significantly the results, but also gives us an insight of the most frequently used RPFs in the collection. Due to space limitations in figure 3 we present only the first 8 RPFs with the biggest appearance rate. Each RPF shown in the figure 3 has an ID number at its left side, and at its right side a number that indicates the times of appearance in the collection. As seen figure 3 most of the RPFs (1,2,3,4,5,7) comprise of simple structures. These structures are expected to be found in a collection of process models as

[2] http://pi.informatik.uni-siegen.de/qudimo/bpmn/
[3] http://www.omg.org/spec/BPMN/20100602/

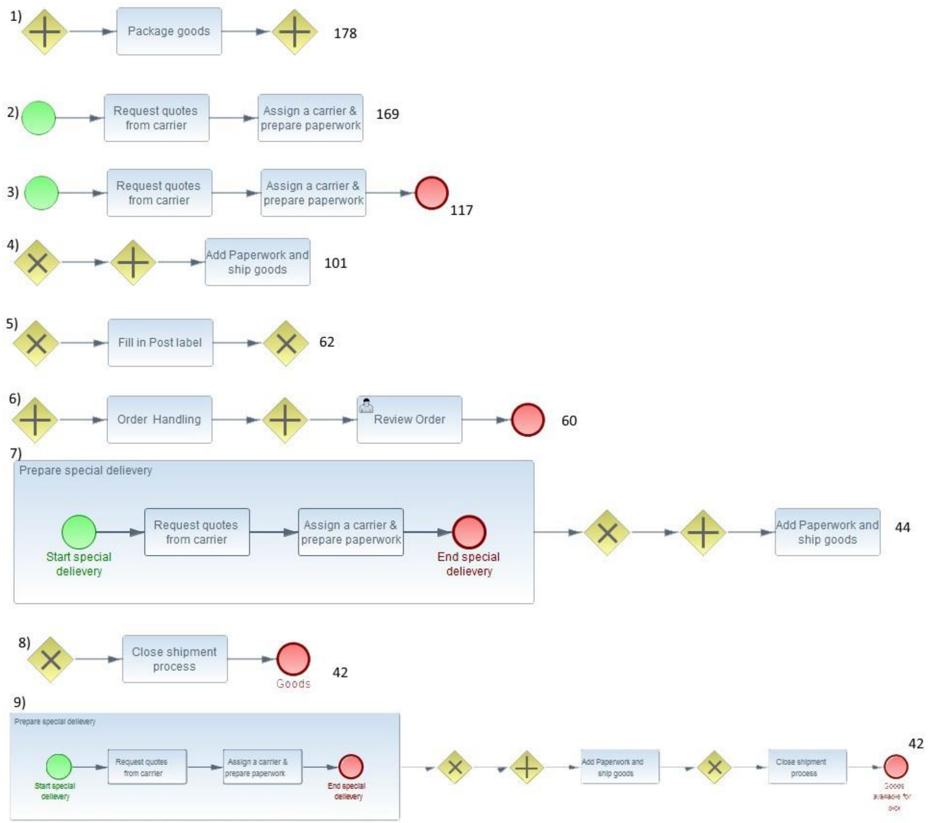
1)
Package goods
178
2)
Request quotes from carrier
Assign a carrier & prepare paperwork
169
3)
Request quotes from carrier
Assign a carrier & prepare paperwork
117
4)
Add Paperwork and ship goods
101
5)
Fill in Post label
62
6)
Order Handling
Review Order
60
7)
Prepare special delievery
Start special delievery
Request quotes from carrier
Assign a carrier & prepare paperwork
End special delievery
Add Paperwork and ship goods
44
8)
Close shipment process
Goods
42
9)
Prepare special delievery
Start special delievery
Request quotes from carrier
Assign a carrier & prepare paperwork
End special delievery
Add Paperwork and ship goods
Close shipment process
42

Figure 3: The first 8 RPFs with the bigger appearance rate

they are simple combinations of parallel and/or exclusive gateways with tasks. Hence, unless someone is interested to learn the appearance ratio of these trivial structures the number of nodes per RPF can be set to a number> 3. Then these RPFs will not be included in the results.

By studying the resulting RPFs it is also possible to draw conclusions about usual practices in the design of process models. For example, RPF 6 reveals that parallel structures are frequently used at the end of process models. It is also interesting to notice that only one branch of the branching structures is found to be repeated each time. This indicates that the rest of the branches followed a different business logic.

General conclusions about our collection can also be drawn. For example, we observe that RPF 7 and RPF 9 are quite big and complex to have so frequent appearances. From this we can conclude many of the analyzed models were different versions of the same business process. As the real analysis will be applied on thousands of real-world process models gathered from different resources, this phenomenon is expected to be eliminated. However, we might have similar phenomena where the RPFs will reveal other details about our collection.

6 Related Work

Process Fragmentation is frequently discussed in the literature, with a focus to Single-Entry-Single-Exit (SESE) regions [18,19]. These regions are then connected together, to form an hierarchy called “Refined Process Structure Tree” (RPST), which is then used to detect these sub-graphs of models that match together [8]. This approach was also examined in our work [17]. However, the division of the model to SESE regions, and then RPSTs would result in a sub-set of fragments, that do not necessarily represent the existing recurring structures in the process models. Therefore, the proposed technique was not considered compliant with our needs. In the field of structural similarities we could not find any approach that focuses on the detection of recurring structural parts in BPMN 2.0 models with a sole focus on their control-flow. A number of algorithms for process model structural matching and comparison are presented in [4,14], but all of them have a strong focus on text semantics to detect similar mappings. As our work focuses on the structure of the process model and is independent of text or behavioral information, it was not possible to further use the aforementioned approaches. In [10] there is a focus on BPEL processes, which are tranformed to a BPEL process tree. Afterwards, tree mining algorithms are applied in order to discover recurring structures in a process. Although the further goal of this work is very similar to our goal, the different nature of BPMN 2.0 language does not allow to apply the same tree mining techniques for similar structures detection.

The Workflow Patterns Initiative[4] is an effort to provide a conceptual basis for process technology [3]. This initiative presents the aspects (control flow, data, resource, and exception handling) that need to be supported by any workflow language. As pattern they describe the minimum sequence of workflow language elements that should be combined to represent a fundamental concept. The proposed micro-structures are not accompanied by information of real-life usage rate. Therefore, we consider this approach to complement but not replace our goals.

[4] http://www.workflowpatterns.com/

7 Conclusion and Future Work

In this work we described an extension of an algorithm defined in [17]. With this extension the algorithm automatically counts the appearances of recurring structures in a collection of process models. The ultimate motivation for this work is the development of a workload generator for benchmarking BPMN 2.0 Workflow Engines. However, the proposed methodology can be also applied to different use-case scenarios for process model re-usability.

To the best of our knowledge, this work is the first attempt to automatically detect frequently used structures in a collection of BPMN 2.0 process models. We have evaluated our approach with a collection of 43 BPMN 2.0 process models. As seen from the resulting RPFs conclusions can be made about a) frequently used structures (usual-practices) in BPMN 2.0 and b) for the collection's special characteristics.

As future work we plan to extend the algorithm for the complete set of BPMN 2.1 model elements. We will run the approach on the complete collection of real-world models, and execute a thorough analysis on the results. As a next step we will implement the first prototype of the process synthesizing methodology.

Acknowledgments This work is funded by the Swiss National Science Foundation and the German Research Foundation with the BenchFlow - A Benchmark for Workflow Management Systems (DACH Grant Nr. 200021E-145062/1) project.

References

1. Basin, D.A.: A term equality problem equivalent to graph isomorphism. Information Processing Letters 51 (1994)
2. Cardoso, J.: Business process control-flow complexity: Metric, evaluation, and validation. International Journal of Web Services Research 5(2), 49–76 (2008)
3. van Der Aalst, W.M.P., Ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distrib. Parallel Databases 14(1), 5–51 (Jul 2003)
4. Dijkman, R., Dumas, M., van Dongen, B., Käärik, R., Mendling, J.: Similarity of business process models: Metrics and evaluation. Inf. Syst. 36(2), 498–516 (Apr 2011)
5. Dijkman, R., Dumas, M., García-Bañuelos, L.: Graph matching algorithms for business process model similarity search. In: Proceedings of BPM '09. pp. 48–63. Springer-Verlag, Berlin, Heidelberg (2009)

6. Eberle, H., Leymann, F., Schleicher, D., Schumm, D., Unger, T.: Process Fragment Composition Operations. In: Proceedings of APSCC 2010. pp. 1–7. IEEE Xplore (December 2010)
7. Eberle, H., Unger, T., Leymann, F.: Process fragments. In: Meersman, R., Dillon, T., Herrero, P. (eds.) On the Move to Meaningful Internet Systems: OTM 2009, Lecture Notes in Computer Science, vol. 5870, pp. 398–405. Springer Berlin Heidelberg (2009)
8. García-Bañuelos, L.: Pattern identification and classification in the translation from bpmn to bpel. In: Meersman, R., Tari, Z. (eds.) OTM Conferences (1). Lecture Notes in Computer Science, vol. 5331, pp. 436–444. Springer (2008)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York, NY, USA (1990)
10. Hertis, M., Juric, M.B.: An empirical analysis of business process execution language usage. IEEE Transactions on Software Engineering 40(8), 1–1 (2014)
11. Jordan, D., Evdemon, J.: Business process model and notation (BPMN) version 2.0. Object Management Group, Inc (January 2011)
12. Ma, Z.: Process fragments: enhancing reuse of process logic in BPEL process models. Ph.D. thesis, Universität Stuttgart (2012)
13. Mendling, J.: Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness. Springer (2008)
14. Pietsch, P., Wenzel, S.: Comparison of bpmn2 diagrams. In: Mendling, J., Weidlich, M. (eds.) Business Process Model and Notation, Lecture Notes in Business Information Processing, vol. 125, pp. 83–97. Springer Berlin Heidelberg (2012)
15. Schumm, D., Leymann, F., Ma, Z., Scheibler, T., Strauch, S.: Integrating Compliance into Business Processes: Process Fragments as Reusable Compliance Controls. In: Schumann/Kolbe/Breitner/Frerichs (ed.) MKWI'10, Göttingen, Germany, February 23-25, 2010. pp. 2125–2137
16. Schumm, D., et al.: Process Fragment Libraries for Easier and Faster Development of Process-based Applications. JSI 2(1), 39–55 (January 2011)
17. Skouradaki, M., Goerlach, K., Hahn, M., Leymann, F.: Application of Sub-Graph Isomorphism to Extract Reoccurring Structures from BPMN 2.0 Process Models. In: SOSE 2015; San Francisco Bay, USA, March 30 - 3, 2015. IEEE (April 2015)
18. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and More Focused Control-Flow Analysis for Business Process Models Through SESE Decomposition. In: Krämer, B., Lin, K., Narasimhan, P. (eds.) Proceedings of ICSOC 2007. Lecture Notes in Computer Science, vol. 4749, pp. 43–55. Springer-Verlag, Berlin (2007)
19. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. In: Dumas, M., Reichert, M., Shan, M.C. (eds.) Business Process Management, Lecture Notes in Computer Science, vol. 5240, pp. 100–115. Springer Berlin Heidelberg (2008)
20. Verma, R.M., Reyner, S.W.: An analysis of a good algorithm for the subtree problem, correlated. SIAM J. Comput. 18(5), 906–908 (Oct 1989)