Representative BPMN 2.0 Process Models Generation from Recurring Structures

Marigianna Skouradaki, Vasilis Andrikopoulos, Frank Leymann

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

BibTeX:

```
@inproceedings {SAL2016,
   author = {Marigianna Skouradaki and Vasilis Andrikopoulos and Frank Leymann},
   title = {{Representative BPMN 2.0 Process Models Generation from Recurring Structures}},
   booktitle = {Proceedings of the 23rd IEEE International Conference on Web Services},
   publisher = {IEEE},
   pages = {468--475},
   month = {June},
   year = {2016},
   keywords = {BPMN 2.0; Business Process Management; collection; composition;
               generation; process model; representative}
}
```

Representative BPMN 2.0 Process Model Generation from Recurring Structures

Marigianna Skouradaki, Vasilios Andrikopoulos, and Frank Leymann
Institute of Architecture of Application Systems (IAAS)
University of Stuttgart, Germany
{skouradaki,andrikopoulos,leymann}@iaas.uni-stuttgart.de

***Abstract*—The use of process fragments to leverage reuse of process models is well established in the literature. Process fragments are manually or semi-automatically extracted and mainly focus on the textual or behavioural semantics of the process models that they are extracted from. However, in many use cases we also need to use these fragments to derive synthetic process models that satisfy specific structural properties. In order to tackle this challenge we propose a method for automatically generating synthetic, representative, executable process models expressed in Business Process Model and Notation 2.0 (BPMN 2.0) with respect to specific user-defined structural criteria. Our method identifies, selects, and combines recurring sub-structures discovered in a collection of thousands of real world process models. The recurring sub-structures are seen as an extended type of process fragments. For our method we have developed a proof-of-concept prototype and for this we discuss the experimental results obtained from its evaluation.**

***Index Terms*—BPMN 2.0, Business Process Management, collection, composition, generation, process model, representative**

I. Introduction

The (semi-)automated generation of synthetic process models is necessary in the absence of available large collections of processes from the industry or academia [18]. In previous work [14], for example, we have discussed automated workload generation as one of the major challenges towards developing a benchmark for Workflow Management Systems. Generation in this context refers to the process of discovering, extracting, selecting, and synthesizing process fragments into models that resemble as much as possible realistic models from practice. Having already discussed the discovery and extraction of fragments in [12], in this work we shift the focus to the selection and synthesis aspects of process model generation for the Business Process Models and Notation 2.0 (BPMN 2.0) language. Our proposed approach is agnostic with respect to its application. This means that we use the benchmarking development problem essentially as a use case for our proposal that, due to its generality, can also be applied for other purposes, e.g. for evaluation of refactoring techniques requiring large repositories of process models [4], [18].

An important issue with respect to process model generation is the *representativeness* of the generated models. In principle, the generated models should have the same set of characteristics as an ideal model collection. The characteristics in question are specific to the purpose and use of the collection, and by extension, that of the generated process model set as well. In order to provide a generic solution, for the purposes of this work we use a minimal set of characteristics that are reusable across use cases: model size, and structural criteria like specific events, e.g. start or end events in the model, number of control/activity nodes, fan-in and -out, etc [3]. We then can informally define what constitutes a *representative generated process model*, or simply representative model for short, as *any model that respects a set of predefined statistical characteristics w.r.t. to its size and structure*. Based on this definition, in the following we present an approach for the automatic generation of representative models.

The contributions of this work can be summarised as follows; we present:

1) a method for automatically generating executable representative generated process models for given sets of structural criteria,
2) a proof-of-concept prototypical implementation of the proposed method, and
3) a qualitative and quantitative evaluation of the proposed approach.

The rest of this paper is structured as follows. Section II discusses the background of the use case adopted as an application of our proposal throughout the paper. Section III provides a high level overview of our proposal in a group of phases to be followed. These phases are discussed more extensively in Section IV. Section V presents the proof-of-concept prototype, based on which we run our evaluation in Section VI. The paper closes with related work (Section VII), and conclusions and future work (Section VIII).

II. Background

The BenchFlow project[1] aims to create the first standard benchmark for BPMN 2.0 compliant Workflow Management Systems. The construction of a robust benchmark lies heavily on the definition of a representative workload model [7]. In other words, the artifacts issued to the system under test during the performance tests should stand for different sets of characteristics. Only then can they reflect the interests of the users that exploit the benchmark results [16]. The workload of our benchmark comprises many artifacts, presented in detail in Ferme et al. [8].

One of the primary artifacts of the benchmark are the process models to be executed by the system under test.

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

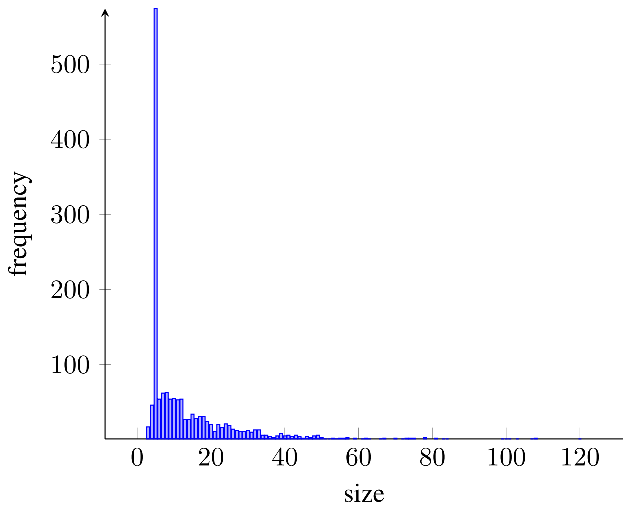

Fig. 1. Frequency of Size in the BPMN 2.0 Models Collection

Towards defining the process models of our workload, we have collected $1,561$ real-world process models that express various application scenarios [14]. Including all these process models to our performance tests would not be feasible or even meaningful. Hence, we need to extract the essence of the collected process models and come up with synthetic process models that reflect the original collection.

Defining the synthetic process models for the workload requires therefore an analysis of the existing process model collection. The purpose of the analysis of our collection is twofold: a) to define a set of business process models properties that will characterize the default workload of the benchmark and b) to enable the synthesis of business process models with respect to specific structural properties defined by the application user. The latter goal is presented in this work. In order to detect the specific properties of the collection, existing business process similarity analysis techniques can be used. Such techniques rely on a) the semantics of the business process model's elements, b) the graph structure of the business process model along with some of its semantics and c) the behavior of the business process model [5].

In the case of BenchFlow, the available collection contains a large amount of anonymized and non-executable (reference) business process models. Therefore, for the analysis of our collection we needed to restraint ourselves on the structural semantics of our models. Fig. 1 shows the size of elements of the existing process models expressed in number of "FlowNodes" [9], hereafter *nodes* per process model and the frequency of appearance for each size. As shown in Fig. 1 most of the models have $size < 20$ while after 20 nodes we observe a much sparser distribution. Furthermore, a good portion of models (around 36.77% of the collection) have exactly 5 nodes ($size = 5$). In addition, the minimum size found in our collection is $min(size) = 3$ while $max(size) = 120$. Respectively, we have $median(size) = 8$ and $mean(size) = 13.91$. Considering the above statistics, for the purposes of this work we have decided that the size of a representative process model should be $5 \leq size \leq 32$, which would cover 87.12% of the collection. For the rest of this discussion we develop the business process models synthesized with respect to these statistics.

III. Method

Our proposed method of process synthesizing is divided into four phases that are presented on Fig. 2. The overall goal of the process synthesizing methodology is to construct a synthetic, executable process model that follows specific structural criteria defined by the application's user.

To this direction the "Characterization" is a preliminary phase that needs to be executed before any request for synthetic process model construction. It takes as input a reference to a database that contains a collection of the recurring structures (Relevant Process Fragments (RPFs)) extracted by the execution of a subgraph isomorphism algorithm like the one presented in [12]. The RPFs are parsed in order to create relevant structural metadata, which are persisted in a separate database. The calculated structural metadata are then considered for constructing the new synthetic process model. This phase can be executed only once for a specific RPF collection, i.e. it is not needed to recreate the metadata database each time the user needs to construct a synthetic process model.

The "Selection" phase chooses the appropriate RPFs with respect to user defined criteria which are given as an input. The criteria contain information about the structural properties of the RPFs to be selected, as well as the size of the new synthetic process model. After selecting the appropriate fragments we proceed to the "Compatibility" phase to apply a compatibility check. This check will examine whether the selected fragments can be linked together to a new business process. This procedure is achieved through a set of rules, which we will discuss in more detail in the following section.

When the compatibility check is finished, the synthesis of the new process model by linking the selected RPFs with each other starts. The synthesized process model is then validated at a first stage against the BPMN 2.0 Standard [9]. This is done with the utilization of external, well-known frameworks such as Camunda BPMN model API [2]. Although the new synthetic business process model might be valid BPMN 2.0 it might not necessarily be representative of real-world use cases. For this reason the generated process model is also validated against the statistical analysis applied to our process model collection, as presented in the previous section. If the validation is not successful the same process is repeated iteratively (starting from the "Selection" phase in order to create a representative model) until a representative model is generated.

As soon as a model is validated successfully, we can then proceed to the serialization of the model as an executable model. Since each Workflow Management System demands a different serialization in order to execute a model [14] this step needs as parameter the specific system which the model

[2]https://github.com/camunda/camunda-bpmn-model

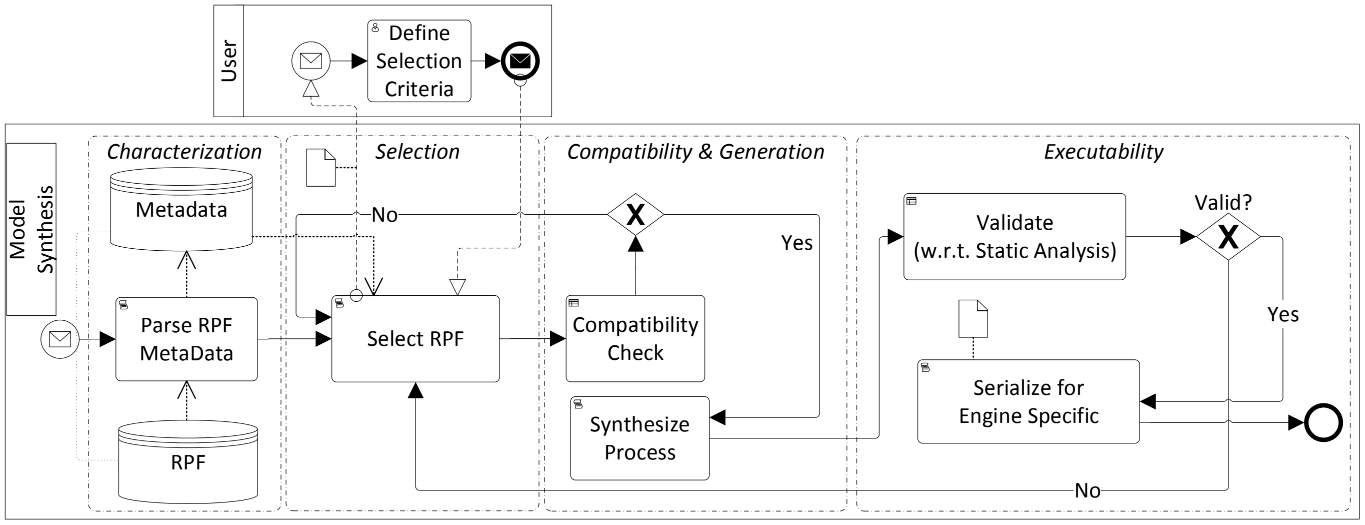

Fig. 2. Method Overview

is meant to be executed in. The executable synthesized process model is then returned to the user.

The following section discusses the phases of our proposed method, as well as the tools and techniques that can be used in more detail.

IV. Representative Model Generation

A. Characterization

The process model generation method is based on the concept of process fragment placeholders as introduced by Schumm et al. [10]. Placeholders are abstract regions in a process fragment which may be replaced by other fragments. While in the original definition [10] the placeholders can be anywhere in the fragment, in our case they can only exist in the beginning and/or at the end of the RPF. More particularly, the placeholders of an RPF are assigned to the positions of the detected sequence flows that are missing a source or a target node [9], hereafter referred to as *open connections*.

With respect to this concept, the goal of this phase is to parse a given collection of RPFs and obtain the structural metadata that characterize the extracted RPFs, as well as to determine the placeholders of the RPFs. For the RPF characterization we have considered the following structural metadata: *hasStartEvent:* indicates if the RPF has a start event; *hasEndEvent:* indicates if the RPF has an end event; *elementsMetadata:* a set of metadata for each detected BPMN 2.0 element in the RPF.

The *elementsMetadata* element contains the following information:

elementGeneralType: information on each task element detected in an RPF (task, gateway, event etc.);

elementSpecificType: information on the specific type of the element (script task, service task, exclusive gateway, parallel gateway etc.);

incomingConnections: the number of open incoming connections that exist on the RPF;

outgoingConnections: the number of open outgoing connections that exist on the RPF.

The *incoming*- and *outgoingConnection* elements are used later towards the identification of the RPFs with open connections and the driving of the process model generation. As discussed in the previous section, the characterization of the RPF collection is executed once for every collection provided. The metadata are then stored in a persistent database, out of which they can be retracted for any future request.

B. Selection

The selection of the RPFs that participate in the generation of the synthetic process model is done according to user-defined criteria. The user-defined criteria define the structural properties that a selected RPF should satisfy. For example the user may either query an RPF by providing general properties (e.g. 2 tasks and 3 gateways) or by requesting exact types of elements (e.g. 4 service tasks and 3 parallel gateways) for each RPF. Each process model generation is also characterized by the expected total number of nodes ($size$) of the new synthetic process model. Overall, the goal of the "Selection" phase is to select RPFs that: a) satisfy the given user-defined criteria, b) can be successfully linked to a complete process model and c) their synthesis will result to a process model with the exact requested size. To this effect, the criteria are combined with size parameters for the evaluation of the appropriate RPF.

Let C be a set of user-defined criteria and R a collection of RPFs which are given as input to our methodology. Then we define the function $\eta : R \rightarrow \mathbb{N} = S$ that calculates the size of each RPF. The function η will map all the elements of the set R to a natural number which corresponds to the process model's size. The result of the function η is the set S.

Let F be a family of sets over S where for each set $N \in F$ it holds: $\{x \in N | \sum_{x \in N} = size\}$ and $|N| = |C|$. In other words, the set F contains all sets of possible combinations of sizes, for which their sum equals to the total requested size of

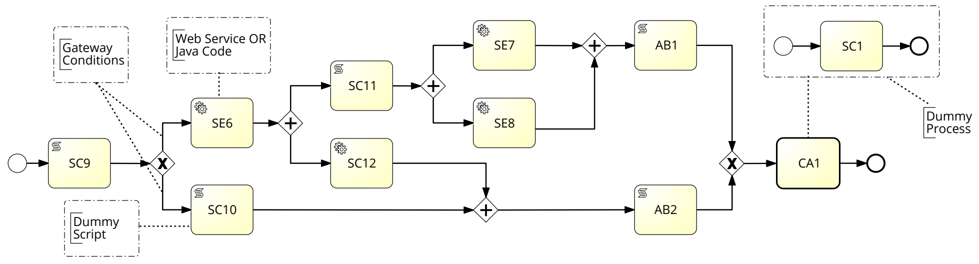

Fig. 3. Generated Synthetic Process Model Example

the new synthetic process model. The number of elements that can be contained in each set of F equals to the total number of user-defined criteria. Let for example $size = 12$, and $|C| = 2$. Then according to the statistics of our collection (cf. Fig 1), $S = [5, 67] \cup [72, 75] \cup \{78, 81, 83, 84, 99, 101, 103, 107, 108\}$. The F is structured as follows: $\{\{5, 7\}, \{6, 6\}, \{7, 5\}\}$.

In order to achieve the size constraints, we need to select RPFs with the appropriate size. The function $\varphi : C \times N \rightarrow R = W_c$ combines a criterion with a size property, to return this subset of the RPF collection that complies with the defined criteria. Lets assume the function φ is applied to the first set F of the previous example. Then $\varphi(1, 5)$ would return all these RPFs with $size = 5$ for the first criterion and $\varphi(2, 7)$ the RPFs with $size = 7$ for the second criterion. The resulting RPFs collections are then given as input to the "Compatibility & Generation" phase that follows.

C. Compatibility & Generation

At this phase we apply a structural compatibility check to determine whether, and which of the selected RPFs can be combined with each other.

Consider the set of L over the selected W sets as they were created in the Selection phase. Then it holds $|L| = |C|$, and the L contains one set of selected RPFs for each criterion. As a preliminary step we remove all the intermediary RPFs that contain start or end events, because a start or an end event in the middle of the synthetic process model is not allowed. Let $\varkappa$ and λ be the predicate functions that indicate if an RPF element contains a start event or an end event respectively. Then $\{\forall x \in W_i, 1 \leq i \leq (|C| - 1)\}$ it needs to hold that $\varkappa(x) = false$ and $\lambda(x) = false$. The next step is to check the open connections between the sets of the returned RPFs and select these RPFs that can be linked to each other. In order to enable the linking between the RPFs we need to verify that there is an equivalent number of open connections between them. Hence, $\forall(W_i, W_{i+1} \in L, 0 \leq i \leq |C| - 1)$ we need to find sequences of pairs (x, y) where $\{\exists\, x \in W_i \wedge \exists\, y \in W_{i+1}, 0 \leq i \leq |C| - 1 \mid x_{outgoingConnections} = y_{incomingConnections}\}$. The above condition is checked for all possible combinations of the resulting sets W until we discover a sequence of $|C|$ RPFs that satisfies it.

In the case we exhaust all possible combinations of RPFs without finding a sequence of RPFs that can be linked the failure of synthesis for these specific criteria is reported. Otherwise, we proceed to the synthesis of the process model by linking the open outgoing with the open incoming connections. At this point, the linking bases solely on the outgoing and incoming connections, without taking into account any particular semantics. At the end, start events are added to the incoming connections of the first RPF, and end events are added to the outgoing connections of the last linked RPF.

Before returning the result of the "Compatibility & Generation" phase, the synthetic process model will be additionally checked against its validity and representativeness. For this purpose, the synthesized process model is checked against the gathered statistics as discussed in Section II, in order to certify that it has a statistical soundness of real-world usage. If any of the aforementioned checks are not satisfied then a corresponding warning or error message is produced and we iteratively select another combination of RPF in order to proceed to the synthesis of a process model.

D. Executability

The process model generated during the previous phase comprises a reference, non-executable model. Each Workflow Management System demands a different serialization of the BPMN 2.0 file in order to execute it [14], and therefore we need to apply the appropriate changes on the generated process model's serialization. Moreover, we need to connect the model elements to the corresponding missing artifacts and external interactions, as they are shown for example in Fig. 3.

The gateway condition is the first element of the process model in Fig. 3 that needs handling. In this case we add a default condition and the appropriate variables to evaluate the control flow path to follow. On the uper path connected to the exclusive gateway exists a service task. We are linking this task to a placeholder "dummy" RESTful web service or a similar Java application. Similarly, on the lower path of the exclusive gateway there is a script task that is linked to a "dummy" (empty) script task. Finally, the process model in Fig. 3 ends with a call activity. The call activity needs to be linked to another external executable process in order to be executed. For this purpose we have created the simple external process as it is shown in Fig. 3.

As seen, our decision for the executable process model is to implement all the external interactions with placeholder activities. Since our process models are generated for benchmarking purposes we need to eliminate the introduced overhead of

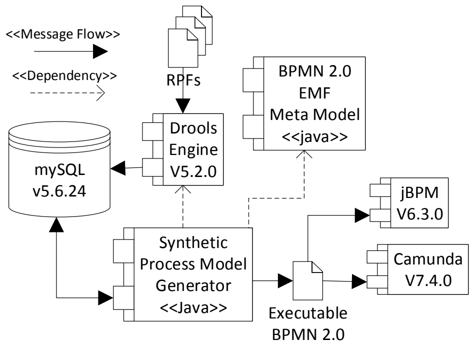

Fig. 4. Architecture of the Process Model Generator Components

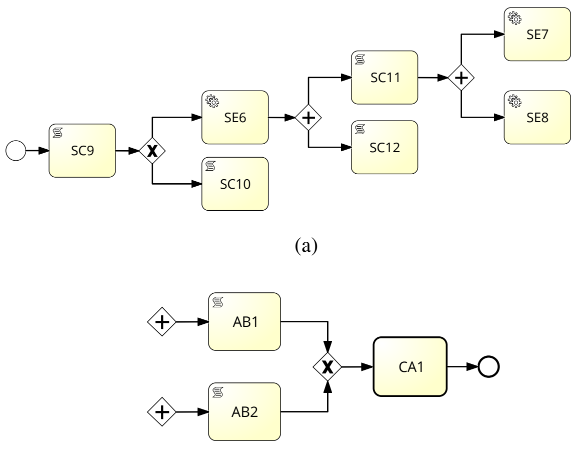

Fig. 5. Selected Relevant Process Fragments

the external interactions. This way we manage to isolate the behavior of the Workflow Managament system to the maximum possible extend and obtain clean measurements [14].

V. Implementation

In order to validate and evaluate the proposed approach we implemented a proof-of-concept prototype that is available in an open source model [3]. A high level architecture of the components of our prototype is shown in Fig. 4. The prototype has been implemented with Java and is based on the BPMN 2.0 EMF metamodel [4]. In addition, the implementation utilizes a MySQL database (v5.6.24) for storing and quering the RPF collection, and a Drools Engine (v5.2.0) for the creation of the RPF structural meta data. The executable process models are tested and validated for the Camunda v7.4.0 [5] engine and the jBPM v6.3.0 [6] engine.

VI. Evaluation

A. Use Case

For the qualitative evaluation of our approach we apply a use case of medium complexity as shown in Fig. 3 (as discussed in Section IV) and Fig. 5. More specifically, the RPFs shown in Fig. 5a and Fig. 5b present the RPFs that were selected by the methodology for the criteria that respond to these of Table I. Namely, for the RPF shown in Fig. 5a we have chosen an RPF with three service tasks, four script tasks, one exclusive gateway, and two parallel gateways. The second RPF shown in Fig. 5b contains two parallel gateways, three script tasks, one call activity and one exclusive gateway.

In this use case, the RPF of Fig. 5a has four open outgoing connections on the tasks: SE7, SE8, SC10, SC12 while the RPF of Fig. 5b expects in total four open incoming connections, two for each of the leftmost parallel gateways. In this case the tasks SE7 and SE8 are connected with the upper parallel gateway and the tasks SC10 and SC12 are connected with the lower parallel gateway. In this way we derive a complete BPMN 2.0 process as it was already shown in Fig. 3. The size of the new synthetic process model is 18.

[3]https://git.io/v2qTG
[4]http://www.eclipse.org/modeling/mdt/?project=bpmn2
[5]https://camunda.org
[6]http://www.jbpm.org

TABLE I
Selection Parameters

	Script Tasks	Service Tasks	Call Activities	Exclusive Gateways	Parallel Gateways
Criterion 1	4	3	0	1	2
Criterion 2	2	0	1	1	2

B. Performance Evaluation

1) Experimental Setup: As this is the first attempt to generate representative and executable BPMN 2.0 process models from a collection of fragments, the quantitative evaluation of our approach could only target this prototype. For the quantitative evaluation of the methodology we use the proof-of-concept prototype discussed in the previous section. For measurement purposes we have used a MacBook Pro with a 3,1 GHz Intel Core i7 and 16 GB memory. On top of it we have set up a Virtual Machine of 2GB memory that runs Ubuntu 64-bit v14.04.3. The applications and frameworks utilized for the prototype are the same as discussed in Section V.

For the setup of the experiments we have used a collection of RPFs, which is derived from the collection of BPMN 2.0 real world process models described in Sec. II. The models are compared through the algorithms described in previous works [12] to detect recurring structures that have at least 5 nodes in common. This limit was set with respect to the properties of our collection, as already discussed. The algorithm discovered and extracted 27,637 RPFs with $size \geq 5$, that were persisted in the RPF database.

We defined three groups of experiments with different numbers of criteria that target different sizes of synthetic process models. The defined criteria are shown in detail in Table II. Before proceeding further we manually queried the RPF database with the above mentioned criteria in order to determine the characteristics of the returned RPFs. By this way we ensure that the defined criteria will result in a large amount of RPF sets, and consequently, that our experiments will stress the prototype performance throughout all the process model generation phases. The main finding derived from our

TABLE II
EXPERIMENTS DETAILS

Experiment	Criteria	Targeted Total Sizes	Criterio 1	Criterio 2	Criterio 3	Criterio 4
1	2	12, 16, 18, 20, 24, 28, 32	CallActivity: 3	CallActivity: 2 ExclusiveGateway: 1	-	-
2	3	18, 20, 24, 28, 32	CallActivity: 3	CallActivity: 2 ExclusiveGateway: 1	CallActivity: 4 ParallelGateway: 2	-
3	4	24, 28, 32	CallActivity: 3	CallActivity: 2 ExclusiveGateway: 1	CallActivity: 4 ParallelGateway: 2	CallActivity: 2 ParallelGateway: 1 ExclusiveGateway: 1

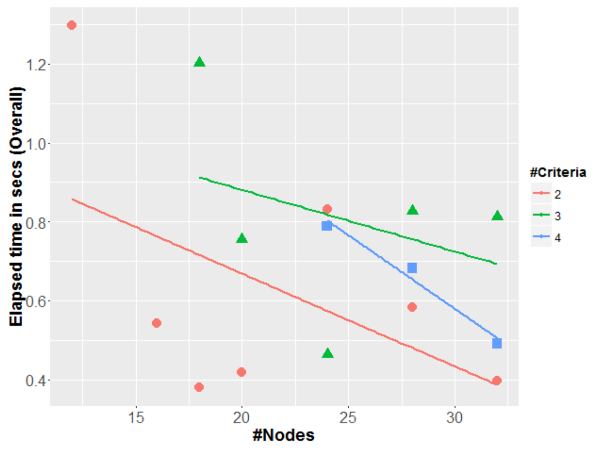

Fig. 6. Time Evaluation of the Synthetic Process Model Generation Per Size

preliminary queries showed that our collection does not contain any script or service tasks. This is justified by the fact that our RPF collection is discovered by process models coming from the same source, and the call activities are in principle preferred for designing process models. On the contrary, there were thousands of RPFs that contained call activities, exclusive and parallel gateways. Hence, at the end we concluded in criteria that target these three types of nodes.

The "Targeted Total Sizes" of our experiments, namely the total number of nodes the generated process model will have is defined with respect to the statistics obtained from our process models collection (cf. Sec. I). Since we need at least two RPFs for the synthesis, the sizes of a representative synthetic model should be $10 \leq size \leq 32$. However, for nodes with $size = 6$ we observe a bigger concentration of process models (cf. Fig. 1), thus we have decided to set the minimum synthesis for $size = 2 \times 6 = 12$.

2) *Experiments Results:* Each experiment was executed 100 times; in each experirement we measured the total amount of time needed for each one of the generation phases, as well as the end-to-end time for the whole generation process. Fig. 6 shows how the size of the generated process models affects time when considering the different criteria in Table II. As seen Fig. 6 the average time for the process model generation is on average 0.5 to 0.8 seconds. What can be observed is that the experiments executed with 4 criteria have a smaller deviation of the results, while the deviation grows proportionally for the experiments of 3 and 2 criteria. Moreover, some outlier points can be observed in Fig. 6. For example, the model generations for the experiments of 2 criteria and for process models of sizes 18 and 20 lasted on average 0.35 and 0.45 seconds respectively. In the first case (size 18) the RPFs of size 6 and 12 nodes are selected from the database. For this query, the database will return 2013 RPFs of 6 nodes and 49 RPF of size 12. As the second set of RPF is small, and we are only requesting a synthesis of 2 RPFs, the combinations can be checked faster, and thus the needed time is smaller. Similarly, in the second case (size 20) an RPF of 6 and an RPF of 14 nodes are selected from the database. Again, in this case RPF size 6 will result in 2013 RPFs while we could only find 1 RPF with 14 nodes in the database. As only one RPF is selected for the second criterion it is anticipated that this experiment would be even faster than the first one. However, in this case the combination of RPFs of 6 and 14 nodes is the second combination to be checked for the generation. The combination for sizes 7 and 13 is the first one to be selected. In this case there are 251 RPFs of size 7 and zero RPFs of size 13. As the size 13 did not return any results, this combination was not further checked for RPFs that can be linked. However, the selection of RPFs for this combination introduces a time overhead. For this reason this experiment lasts in average slightly more than the first case.

Concerning the experiments that needed more time than the average, we have again two outliers. The experiment of 2 criteria and size 12 needs 1.3 seconds to be completed, and the experiment with 3 criteria and 18 nodes needs 1.2 seconds for its completion. In the first case the sizes combination that leads to the process model generation is of 2 RPFs of size 6, and this is the second attempted size combination. As discussed before, the size 6 will return 2013 RPFs from the database. The failure of the first sizes combination, and the large amount of RPFs returned by the second combination result in this experiment being more time-consuming than the rest. The second outlier succeeds in the generation of a process model already from the first attempted size combination. However, the combination that leads to a generated process model is of RPFs with sizes 5,6 and 7. The database has the biggest population for the sizes 5 and 6 and returns 1480, 2013, and 251 RPFs respectively. The big amount of combinations that needed to be checked in this case before finding the RPFs that can be synthesized costs more time than the rest of the experiments.

In Fig. 7 we present the overall average time needed for the experiments as well as the average times needed for each

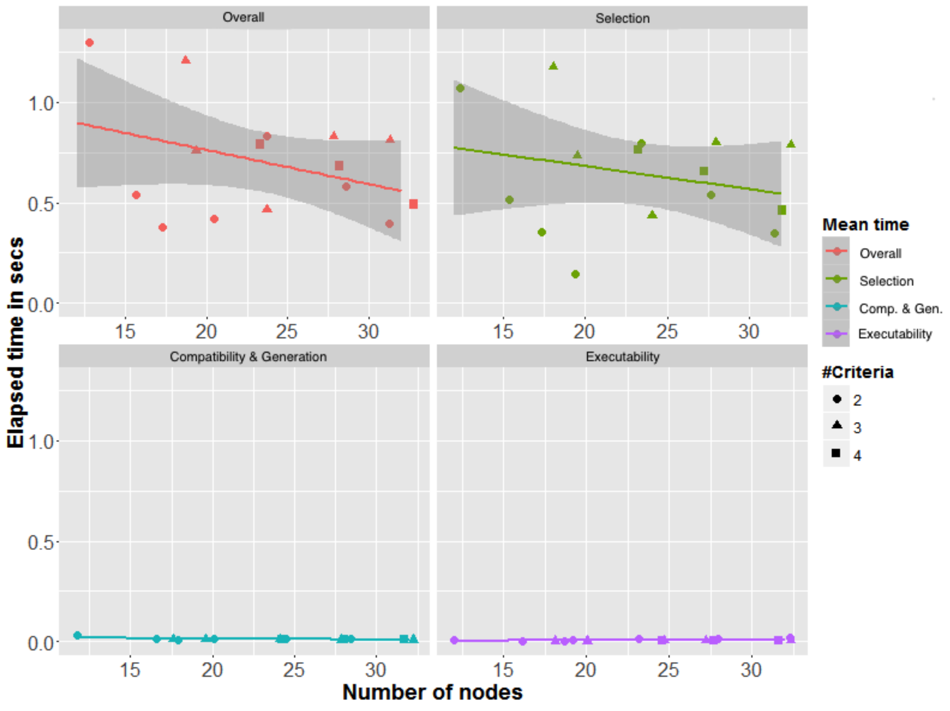

Fig. 7. Time Evaluation of the Synthetic Process Model Generation Per Phase

phase of the process model generation. As observed from Fig. 7 the overall average time drops while the total size of the generated process model increases. In more detail, the phases of “Compatibility & Generation” and “Executability” are basically executed in constant time, independently from the size of the generated process model, and lean towards 0. On the contrary the “Selection” phase is the most time consuming phase and affects heavily the overall time of the process model generation process. The “Selection” phase contains the query to the database in order to select the RPFs that satisfy the user defined criteria. So, the observed times are basically the times needed for the query execution.

It emerges from the results that the “Selection” phase affects the overall time of the process model generation. However, the data points that form the “Overall Time” chart differ slightly from the “Selection” chart’s data points. This implies that there are more factors affecting the process model generation and they need to be further investigated.

3) Discussion: In general, it can be concluded that our evaluation shows realistic times for the generation of representative models. In average, the generation time of a representative process model is on average 0.5 seconds, while the slowest experiment lasted 1.2 seconds. As shown by Fig. 7, the database query is the main influential factor on the overall performance. Thus an optimization of the query, or another choice of database could lead to even better results. The phases of “Compatibility & Generation” and “Executability” are constant and take times that are almost zero. The database query is further influenced by the content of the database. In the investigated use case, the database contained more RPFs with lower sizes ($[5-8]$) and less RPFs with bigger sizes ($[9-15]$). This leads into querying less entries in the database for generating models of bigger sizes, and thus gaining the results faster. Consequently, the overall time drops while the size of the generated process model increases.

Concerning the number of size combinations participating in each experiment, in most of the cases the first or the second size combination leads to the generation of a process model. Generally, the failure of a size combination to provide RPFs that can be synthesized introduced a small time overhead. Currently, the size combinations to be attempted are chosen in a random way. Although we do not have an extensive evaluation on the impact of the chosen size combinations to the overall generation times, our evaluation shows that the choice of the combination can be optimized with respect to the content of the database. So, for example, if we know that a database contains less RPFs for bigger sizes, we should first attempt the size combinations of bigger sizes. This way the time of the “Selection” phase will drop and as a result the overall composition will be faster. Further optimization is the goal of future work.

VII. Related Work

The concept of using process fragments as reusable elements to compose new process models is frequently discussed in the literature [6], [10], [11], [17], [19]. Schumm et al. [10], [11] introduce the concept of process fragments libraries that enable the easier and faster development of process models. We are utilizing this concept for storing and querying the RPFs, which are an extended type of process fragments. The above approach bases on the textual semantics of the stored process fragments. Our approach focuses on the structural characteristics of the RPFs. Eberle et al. [6] present a formal model for process fragments and corresponding operations for their composition. In this work, we adopt the suggested methods for composing the process fragments into process models, and we have extended the composition function in order to stress the representativeness of the generated process model.

Yan et al. [18] propose a complete method for generating synthetic process models. The authors propose the usage of refined process structure tree (RPST) [15] as a possible improvement of their work. In previous work, we have argued that RPST cannot be utilized for these purposes, and we have therefore introduced the concept of PRFs [12]. Hence, this work can be considered as an extension to the work of Yan et al. [18].

To the best of our knowledge, this is the first effort to generate representative and executable BPMN 2.0 process models with respect to structural characteristics. Graph synthesis can be considered the most related area to our approach. Akoglu et al. [1] propose an method to recursively generate realistic graphs through a random typing graph generator. Bader and Madduri [2] summarize three approaches of graph generators. One of these approaches (SSCA#2 graph generator) aims to produce graphs that are in turn used for benchmarking purposes. Although the application field and method differs, this approach has the same motivation goal as ours.

On the whole, the aforementioned works have different application fields. Our work focuses on the synthetic graph generation for BPMN 2.0 business process models. In addition we support the serialization of the synthetic BPMN 2.0 process model to its executable equivalent for different Workflow Management Systems.

VIII. Conclusion and Future Work

In this work, we introduced a method to automatically generate synthetic BPMN 2.0 process models with respect to their structural characteristics. The generated process models are also automatically serialized to their executable equivalent for different BPMN 2.0 Workflow Management Systems. The method can be utilized for the creation of single process models, or even collections of thousands of synthetic representative process models. We implemented a proof-of-concept prototype of our proposal, which we evaluated through a set of use cases and experiments. The evaluation has shown that our method runs in acceptable, realistic times, with our longer lasting experiment reaching 1.2 seconds of execution. The overall performance of the proposed method is heavily influenced by the execution of the Selection phase, because querying and selecting thousands of process models is in principle more expensive than the in-memory computations of the other phases. Our results also indicate that the overall process model generation is influenced by the composition of the extracted fragment metadata. As a general observation, the overall time is inversely proportional to the number of RPFs that satisfy a specific criterion.

In the future we plan to optimize the overall time needed by further optimizations on the database, and to conduct an evaluation of the impact of chosen size combinations to the overall process model generation times. Concerning the overall method we intend to extend it by making the models more consistent by extending the method for a larger non-anonymized collection to consider also the textual semantics of the process models; by simulating probabilistic executions and imitating representative behaviors and by including only the most frequently occurring RPFs in the generation process [13].

Acknowledgements

The authors would like to thank A. Wahab for the contribution to the prototypical implementation. This work is funded by the project DACH Grant Nr. 200021E-145062/1.

References

[1] L. Akoglu and C. Faloutsos. *Proc. ECML PKDD'09, Part I*, chapter RTG: A Recursive Realistic Graph Generator Using Random Typing, pages 13–28. Springer Berlin Heidelberg, 2009.

[2] D. A. Bader and K. Madduri. Gtgraph: A synthetic graph generator suite, 2006.

[3] J. Cardoso. Business process control-flow complexity: Metric, evaluation, and validation. *IJWSR*, 5(2):49–76, 2008.

[4] R. Dijkman, M. la Rosa, and H. A. Reijers. Editorial: Managing large collections of business process models-current techniques and challenges. *Comput. Ind.*, 63(2):91–97, Feb. 2012.

[5] M. Dumas, L. García-Bañuelos, and R. M. Dijkman. Similarity search of business process models. *IEEE Data Eng. Bull.*, 32(3):23–28, 2009.

[6] H. Eberle, F. Leymann, et al. Process fragment composition operations. In *Proc. APSCC' 2010*, pages 157–163, Dec 2010.

[7] D. G. Feitelson. *Workload modeling for computer systems performance evaluation*. Cambridge University Press, 2015.

[8] V. Ferme, A. Ivanchikj, et al. A container-centric methodology for benchmarking workflow management systems. In *Proc. CLOSER '16*. SciTePress, April 2016. to appear.

[9] D. Jordan and J. Evdemon. Business Process Model And Notation (BPMN) Version 2.0. Object Management Group, Inc, January 2011. http://www.omg.org/spec/BPMN/2.0/.

[10] D. Schumm, D. Karastoyanova, et al. Process fragment libraries for easier and faster development of process-based applications. *Journal of Systems Integration*, 2(1):39–55, 2011.

[11] D. Schumm, F. Leymann, et al. Integrating compliance into business processes: Process fragments as reusable compliance controls. In *Proc. MKWI '10*. Universitätsverlag Göttingen, 2010.

[12] M. Skouradaki, Görlach, et al. In *Proc. SOSE '15*, San Francisco Bay, CA, USA, March 30 – April 3 2015. IEEE Computer Society.

[13] M. Skouradaki and F. Leymann. Detecting frequently recurring structures in bpmn 2.0 process models. In *Proc. SummerSOC'15*, pages 102–116. IBM, 2015.

[14] M. Skouradaki, D. H. Roller, et al. On the road to benchmarking BPMN 2.0 workflow engines. In *Proc. ICPE '15*, pages 301–304, 2015.

[15] J. Vanhatalo, H. Völzer, and J. Koehler. The refined process structure tree. In *Proceedings of the 6th International Conference on Business Process Management*, BPM '08, pages 100–115, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] J. von Kistowski, J. A. Arnold, et al. How to build a benchmark. In L. K. John, C. U. Smith, K. Sachs, and C. M. Lladó, editors, *Proc. ICPE '15*, pages 333–336. ACM, January 2015.

[17] A. Weiß, V. Andrikopoulos, et al. Enabling the Extraction and Insertion of Reusable Choreography Fragments. In *Proc. ICWS '15*, pages 686–694. IEEE Computer Society, June 2015.

[18] Z. Yan, R. Dijkman, and P. Grefen. Generating process model collections. *Software & Systems Modeling*, pages 1–17, 2015.

[19] R. Yang, B. Li, et al. Scky: A method for reusing service process fragments. In *Proc. ICWS '14*, pages 209–216. IEEE Computer Society, June 2014.