Generating process model collections

Citation for published version (APA):

Yan, Z., Dijkman, R. M., & Grefen, P. W. P. J. (2017). Generating process model collections. *Software and Systems Modeling*, 16(4), 979-995. <https://doi.org/10.1007/s10270-015-0497-6>

Document license:

TAVERNE

DOI:

[10.1007/s10270-015-0497-6](https://doi.org/10.1007/s10270-015-0497-6)

Document status and date:

Published: 01/10/2017

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Generating process model collections

Zhiqiang Yan^{1,2} · Remco Dijkman³ · Paul Grefen³

Received: 22 November 2014 / Revised: 1 September 2015 / Accepted: 5 September 2015 / Published online: 13 October 2015
© Springer-Verlag Berlin Heidelberg 2015

Abstract Business process management plays an important role in the management of organizations. More and more organizations describe their operations as business processes. It is common for organizations to have collections of thousands of business processes, but for reasons of confidentiality these collections are often not, or only partially, available to researchers. On the other hand, research on techniques for managing process model collections, such as techniques for process retrieval, requires large collections for evaluation purposes. Therefore, this paper proposes a technique to generate such collections of process models, based on the properties of real-world collections. Where existing techniques focus on the structure of the process models, the technique proposed in this paper also generates task labels that consists of words from real-life task labels and considers semantic information of node and edge types. We evaluate our technique by applying it to generate two synthetic collections of process models of over 60,000 and over 2,000 models, respectively. We show that the generated synthetic collections have similar properties to the original collections. To the best of our knowledge, this is the first technique that

can generate synthetic BPMN models, thus enabling experimentation with process collections that have laboratory-set quantitative parameters and qualitative properties that are based on real-world process model collections.

Keywords Synthetic process models · Process model generation · Process model similarity

1 Introduction

Nowadays, business process management technologies are developing quickly in both academia and industry [8]. As a result, it is common to see collections of hundreds or even thousands of business process models. For example, the SAP reference model consists of 604 business process models [6], and the information department of China Mobile Communication Corporation (CMCC) maintains more than 8000 processes [9]. To manage such large numbers of process models efficiently, business process model repositories are required [16,29]. Such repositories provide techniques for storage management [17], searching a collection for processes that are similar to a given process [10,12,13,30], querying a collection to find models with certain properties [22,31], identifying differences between process models [18,26,27,33], and refactoring a collection of process models [7].

The evaluation of such techniques requires that they are applied to large collections of real-life business process models. However, these models are the property of companies and are usually not made publicly available for reasons of confidentiality. Consequently, it is difficult for researchers to obtain a large amount of real-life process models to evaluate their techniques. To solve the disconnection between the availability of large collections of models from practice and the necessity to have such collections for evaluation purposes,

Communicated by Dr. Alan Hartman.

✉ Zhiqiang Yan
zhiqiang.yan.1983@gmail.com

Remco Dijkman
r.m.dijkman@tue.nl

Paul Grefen
p.w.p.j.grefen@tue.nl

¹ Capital University of Economics and Business, Beijing 100070, China

² Tsinghua University, Beijing 100080, China

³ Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven, The Netherlands

Fig. 1 A collection of business process models and the corresponding process graphs

All nodes have the type 'task', except for the parallel gateway in graph 4.
All edges have the type 'sequence flow'.

related areas of research often resort to the generation of synthetic collections of models. For example, in the graph database area, synthetic graph generation techniques exist [14].

Therefore, this paper proposes a technique that generates large collections of process models with real-world properties. The properties that are considered for model generation include the size of the process models, the frequency of the occurrence of common workflow patterns (sequences, splits, joins, and loops), the words in the labels, the size of the labels, the co-occurrence of words, and the types of nodes and edges. The steps that are proposed for generating process models are as follows. First, the properties of the process models are extracted from a given collection. Second, labels and workflow patterns are generated based on the extracted properties. Third, a new synthetic model is generated by repeatedly inserting a (generated) pattern, until the size of the synthetic model reaches a preselected number. Finally, the types of the nodes and the edges (e.g., intermediate event and manual task) are assigned.

Against this background, the rest of the paper is organized as follows. After presenting preliminaries in Sects. 2, 3 explains how properties are extracted from a given process model collection. Sections 4 and 5 present algorithms to generate synthetic process models and the experiments to evaluate the algorithms, respectively. Section 7 presents the conclusions.

2 Business process graph

We define our technique on (business) process graphs.¹ A process graph is a graph-based representation of a process

¹ The definition of process graph is extended compared to the definition in [32] to capture node and edge types (μ and ν).

model. The benefit of using process graphs is twofold. First, it provides an easy formalism to work with. While the actual BPMN metamodel [21] is quite complex with many syntactical elements, process graphs are more compact; the core concepts are 'node' and 'edge,' and variations in syntactical elements are represented as attributes of nodes or edges. This makes the definition of the algorithms much simpler. Second, process graphs can be generated for models that are constructed with different business process modeling languages. This makes our algorithms more flexible. For example, in the evaluation, we used both EPC models and BPMN models as input to generate synthetic BPMN models.

Definition 1 (*Process graph, pre-set, post-set*) Let $\mathcal{L}$ be a set of labels, $\mathcal{T}_N$ be a finite set of node types, and $\mathcal{T}_E$ be a finite set of edge types. A process graph describes a (business) process as a tuple $(N, E, \lambda, \mu, \nu)$, in which:

- N is the finite set of nodes.
- $E \subseteq N \times N$ is the finite set of edges.
- $\lambda : N \rightarrow \mathcal{L}$ is a function that maps nodes to labels.
- $\mu : N \rightarrow \mathcal{T}_N$ is a function that maps nodes to node types.
- $\nu : E \rightarrow \mathcal{T}_E$ is a function that maps edges to edge types.

Let $G = (N, E, \lambda, \mu, \nu)$ be a process graph and $n \in N$ be a node: $\bullet n = \{m | (m, n) \in E\}$ is the pre-set of n , while $n \bullet = \{m | (n, m) \in E\}$ is the post-set of n .

As an example, Fig. 1 shows five BPMN models and their corresponding business process graphs. As shown in graph 4 of Fig. 1, we assign each gateway node a unique label to represent its routing function (e.g., 'ParallelGateway'). Figure 2 shows a BPMN model² with more different node types (e.g., start/end event, user task, and exclusive gateway).

² The model is taken from <http://www.omg.org/spec/BPMN/>.

Fig. 2 A BPMN model and the corresponding process graph

3 Properties of business process model collections

This section presents properties that are extracted from business process model collections. Three different kinds of properties are considered: label properties, control flow properties, and type properties. For labels, properties of words in labels are extracted. For control flows, properties of structural features of process models are extracted. For types, properties of node and edge types are extracted.

3.1 Label properties

Labels consist of words. Therefore, to generate synthetic labels with similar properties to labels in a collection, we consider two label properties: the frequency of occurrence of a word and the frequency of the co-occurrence of two words. The former is the frequency and probability of a word appearing in a synthetic label. The latter is the frequency and probability of two words appearing in a synthetic label, or in the labels of two connected nodes. In the remainder of this paper, we will consider synthetic labels as a bag of words, in which the words are taken from the labels of the original models. Synthetic labels are subsequently generated by combining these words, based on the frequency and probability of their (co-)occurrence. More advanced techniques to generate labels can be envisioned. Such techniques are especially helpful to facilitate the evaluation of algorithms that perform more operations on the semantics of labels (e.g., [20,28]). We leave this for future work.

To facilitate the processing of labels, we use lowercase versions of the words and remove stop words (e.g., 'a', 'an', 'the', 'one', ...) and gateway labels (e.g., 'ParallelGateway').

Definition 2 defines the word set of a process model collection.

Definition 2 (*Word set, label size*) Let $\mathcal{L}$ be the label set of a collection of process graphs with disjoint sets of nodes. The function $\omega(l)$ maps a label l to the set of words that appear in l .

The word set $\mathcal{W}$ of the collection of process graphs is the set of words that appear in $\mathcal{L}$. Formally, $\mathcal{W} = \{w | w \in \omega(l) \wedge l \in \mathcal{L}\}$.

The size of a label l is the number of words in l , i.e., $|\omega(l)|$.

For example, the word set of the collection in Fig. 1 is $\{\text{order, goods, receive, online, pay, application, and approve}\}$.

To generate a label, we must select the words that should be part of that label. To make that selection, the probability of word occurrence is required. For a word in the word set of a collection, the frequency and the probability of its occurrence are defined in as follows.

Definition 3 (*Frequency of word occurrence, probability of word occurrence*) Let $\mathcal{N}$ be the node set of a collection of process graphs with disjoint sets of nodes, and $\mathcal{W}$ be the word set of that collection.

The frequency of the occurrence of a word w , denoted as $FWO(w)$, is the number of nodes in the collection that contain the word w in their labels. Formally, $FWO(w) = |\{n \in \mathcal{N} | w \in \omega(\lambda(n))\}|$.

The probability of the occurrence of a word w is the frequency of the occurrence of w divided by the frequency of the occurrence of all words. Formally, $PWO(w) = \frac{FWO(w)}{\sum_{w_1 \in \mathcal{W}} FWO(w_1)}$.

For example, in Fig. 1, the frequency of the word occurrence of 'goods' is 8, and its probability is $8/22 = 0.36$.

To select a word that should be part of a synthetic label, we can also take other words that appear in that label into account. We can even take words into account that appear in the labels of connected nodes. To that end, three types of word co-occurrence are considered as defined in Definition 4, i.e., word co-occurrence, pre-word co-occurrence, and post-word co-occurrences. Gateway nodes are not considered for pre-word (post-word) co-occurrence. For a node n , if a node $n_1 \in \bullet n$ ($n \bullet$) is a gateway node, nodes in $\bullet n_1$ ($n_1 \bullet$) of the gateway nodes are considered instead of n_1 . For example, in graph 4 of Fig. 1, the post-word co-occurrence for words in the node 'order goods', nodes 'receive goods' and 'pay' are considered instead of the gateway node 'ParallelGateway'. Consequently, the three types of word co-occurrence are defined as follows.

Definition 4 (*Frequency of word co-occurrence*) Let $\mathcal{N}$ be the node set of a collection of process graphs with disjoint sets of nodes and $\omega(l)$ be the function that maps a label l to the set of words that appear in l .

- Frequency of word co-occurrence ($FWCO$): Given a word w and another word w_1 ($w_1 \neq w$), the frequency of the word co-occurrence of w and w_1 is the number of nodes of the collection of process graphs that contain both w and w_1 . Formally, $FWCO(w, w_1) = |\{n \in \mathcal{N} | w, w_1 \in \omega(\lambda(n))\}|$. We say that word w_1 co-occurs in the same node label with w , if $FWCO(w, w_1) > 0$.
- Frequency of Pre-Word Co-Occurrence ($FWCO_{pre}$): Given two words w and w_1 , the frequency of the pre-word co-occurrence of w_1 with respect to w is the number of process fragments (a sequence of two nodes) satisfying that w appears in the label of a node n , w_1 appears in the label of a node n_1 , and there exists a process graph g , in which n_1 is in the pre-set of n . Formally, let $\mathcal{D}$ be a set of process graphs with disjoint sets of nodes $FWCO_{pre}(w, w_1) = |\{n_1 \in \mathcal{N} | g \in \mathcal{D} \wedge n, n_1 \in N_g \wedge n_1 \in \bullet n \wedge w \in \omega(\lambda(n)) \wedge w_1 \in \omega(\lambda(n_1))\}|$. We say that word w_1 co-occurs with w in a pre-set node label, if $FWCO_{pre}(w, w_1) > 0$.
- Frequency of Post-Word Co-Occurrence ($FWCO_{post}$): Given two word w and w_1 , the frequency of the post-word co-occurrence of w_1 with respect to w is the number of process fragments (a sequence of two nodes) that satisfy that w appears in the label of a node n , w_1 appears in the label of a node n_1 , and there exists a process graph g , in which n_1 is in the post-set of n . Formally, let $\mathcal{D}$ be a set of process graphs with disjoint sets of nodes $FWCO_{post}(w, w_1) = |\{n_1 \in \mathcal{N} | g \in \mathcal{D} \wedge n, n_1 \in N_g \wedge n_1 \in n \bullet \wedge w \in \omega(\lambda(n)) \wedge w_1 \in \omega(\lambda(n_1))\}|$. We say

that word w_1 co-occurs with w in a post-set node label, if $FWCO_{post}(w, w_1) > 0$.

For example, with respect to the word 'receive' in the collection in Fig. 1, the frequency of the word co-occurrence for 'goods' is 4, the frequency of pre-word co-occurrence for 'goods' is 4, and the frequency of post-word co-occurrence for 'goods' is 1.

The probabilities of the three types of word co-occurrence are defined in Definition 5.

Definition 5 (*Probability of word co-occurrence*) Let $\mathcal{N}$ be the node set of a collection of process graphs with disjoint sets of nodes. Furthermore, let $\mathcal{W}$ be the word set, and $\omega(l)$ the function that maps a label l to the set of words that appear in l . Three types of probability of word co-occurrence are defined as follows.

- Probability of word co-occurrence ($PWCO$): Given a word w and another word w_1 , the probability of the word co-occurrence of w and w_1 is the frequency of w_1 co-occurring with w divided by the frequency of all words co-occurring with w . Formally,

$$PWCO(w, w_1) = \frac{FWCO(w, w_1)}{\sum_{w_2 \in \mathcal{W} \wedge w_2 \neq w} FWCO(w, w_2)}. \quad (1)$$

- Probability of Pre-Word Co-Occurrence ($PWCO_{pre}$): Given two words w and w_1 , the probability of the pre-word co-occurrence of w_1 and w is the frequency of w_1 co-occurring with w in a pre-set node label divided by the frequency of all words co-occurring with w in a pre-set node label. Formally,

$$PWCO_{pre}(w, w_1) = \frac{FWCO_{pre}(w, w_1)}{\sum_{w_2 \in \mathcal{W}} FWCO_{pre}(w, w_2)}. \quad (2)$$

- Probability of Post-Word Co-Occurrence ($PWCO_{post}$): Given two words w and w_1 , the probability of the post-word co-occurrence of w_1 and w is the frequency of w_1 co-occurring with w in a post-set node label divided by the frequency of all words co-occurring with w in a post-set node label. Formally,

$$PWCO_{post}(w, w_1) = \frac{FWCO_{post}(w, w_1)}{\sum_{w_2 \in \mathcal{W}} FWCO_{post}(w, w_2)}. \quad (3)$$

For example, for the word 'receive' in the collection in Fig. 1, the probability of the word co-occurrence with 'goods' is $4/5 = 0.80$, the probability of the pre-word co-occurrence with 'goods' is $4/9 = 0.44$, and the probability of the post-word co-occurrence with 'goods' is $1/3 = 0.33$.

3.2 Structural properties

The structure of a process model can be described in terms of a composition of common workflow patterns, which are called features in this paper. In this section, the most common workflow patterns are considered: sequence, split, join, and loop. These features are used to compose synthetic graphs later on. Properties of these features are considered structural properties. For example, the feature ‘type’ and the feature ‘size’ are considered structural properties as defined in Definition 6. Furthermore, the rules for composing structural features are extracted, which indicate how features can be composed to form a process graph.

Definition 6 (*Basic feature*) A feature f of a process graph is a subgraph of that process graph. The size of a feature is the number of edges that it contains, denoted as $Size(f) = |E_f|$. Let max be a threshold, indicating the maximum size of a feature that is considered. The type of a feature, denoted $Type(f)$, is the structural pattern of that feature $Type(f) \in \{node, sequence, split, join, loop\}$. Feature f is

- a sequence feature of size $s - 1$ consisting of nodes $\{n_1, n_2, \dots, n_s\}$, if E_f is the minimal set containing $(n_1, n_2), (n_2, n_3), \dots, (n_{s-1}, n_s)$, for $s \geq 2$. A sequence feature is denoted as $n_1 \rightarrow n_2 \rightarrow \dots \rightarrow n_s$.
- a split feature of size s consisting of a split node n and a set of nodes $\{n_1, n_2, \dots, n_s\}$, if and only if E_f is the minimal set containing $(n, n_1), (n, n_2), \dots, (n, n_s)$, for $s \geq 2$. A split feature is denoted as $n \rightarrow \{n_1, n_2, \dots, n_s\}$.
- a join feature of size s consisting of a join node n and a set of nodes $\{n_1, n_2, \dots, n_s\}$, if and only if E_f is the minimal set containing $(n_1, n), (n_2, n), \dots, (n_s, n)$, for $s \geq 2$. A join feature is denoted as $\{n_1, n_2, \dots, n_s\} \rightarrow n$.
- a loop feature of size s consisting of nodes $\{n_1, n_2, n_3, \dots, n_s\}$, if E_f is the minimal set containing $(n_1, n_2), (n_2, n_3), \dots, (n_{s-1}, n_s)$, and (n_s, n_1) , for $s \geq 1$. A loop feature is denoted as $n_1 \rightarrow n_2 \rightarrow \dots \rightarrow n_{s-1} \rightarrow n_s$.

The sequence, split, join, and loop features are referred as structural features.

For example, for *graph 4* in Fig. 1, the set of the basic sequence features of size 1 consists of the sequences ‘Order Goods’ $\rightarrow$ ‘ParallelGateway’, ‘ParallelGateway’ $\rightarrow$ ‘Receive Goods’, ‘ParallelGateway’ $\rightarrow$ ‘Pay’, and ‘Receive Goods’ $\rightarrow$ ‘Order Goods’, the basic split feature with split node ‘ParallelGateway’ and the set of nodes ‘Receive Goods’, ‘Pay’, and the basic loop feature with the three basic nodes ‘Order Goods’, ‘ParallelGateway’, and ‘Receive Goods’.

A feature may be used to generate other features, which are called child features.

Definition 7 (*Parent feature, child feature*) If a feature f can generate feature cf by adding a single edge and at most one node, feature f is a direct parent feature of feature cf and feature cf is a direct child feature of feature f . This is denoted as $f \in DPFS(cf)$ or $cf \in DCFS(f)$, where $DPFS(DCFS)$ is a function that maps a feature to its direct parent (child) feature set. The parent and the child relation are the transitive closure of the direct parent and the direct child relation.

For example, for *graph 4* in Fig. 1, the direct child feature set of the node feature ‘Receive Goods’ is the set that consists of the sequence features ‘ParallelGateway’ $\rightarrow$ ‘Receive Goods’ and ‘Receive Goods’ $\rightarrow$ ‘Order Goods’.

As defined in Definition 6, four types of structural features are considered, i.e., sequence, split, join, and loop. Process graphs consist of compositions of these features. For example, *graph 6* of Fig. 3 consists of a sequence feature $a \rightarrow b \rightarrow c$, a split feature $b \rightarrow \{c, d, e, f, g\}$, a join pattern $\{c, d\} \rightarrow h$, a loop feature $f \rightarrow k$, etc.

When splitting up a process graph into features, a feature can be a parent of different features. For example, in *graph 6* of Fig. 3, the sequence feature $b \rightarrow c$ is a parent of the sequence features $a \rightarrow b \rightarrow c \rightarrow h$, $b \rightarrow c \rightarrow h$, etc. This affects the probability of the occurrence of a certain type of feature. For example, sequence features may be counted too often, because all structural features consist of sequence features of size 1. Therefore, only local maximum features are considered. A local maximum feature is a feature without child features, as defined in Definition 8. For example, sequence feature $b \rightarrow c$ is not considered any more, while sequence feature $a \rightarrow b \rightarrow c \rightarrow h$ is considered.

Definition 8 (*Local maximum feature*) Let $g = (N, E, \lambda)$ be a business process graph. Let f , a subgraph of g , be a feature. The feature f is a local maximum feature of g , denoted as $f \in LMF(g)$, if and only if there is not another feature f_1 of g , such that f is a parent feature of f_1 .

In a process graph, a node can be in different local maximum features and have different pre-sets or post-sets in these features. These nodes are called open nodes. In Sect. 4, when generating process graphs, we use these nodes as the points where we extend a process graph and create a larger synthetic graph. An open node is defined as follows.

Definition 9 (*Open node, closed node*) Let g be a process graph and f be a feature of g . A node n of the feature f is an open node for that feature, if and only if the feature does not contain all of the nodes in the pre-set or post-set of n . Formally, $\forall n \in N_f : n \in ON(f) \iff \exists((\bullet n \cup n \bullet) - N_f) \neq \emptyset$. A node n in the feature is an pre-open (post-open) node for the feature, if and only if the feature does not contain all of the nodes in the pre-set (post-set) of n , denoted

Fig. 3 A business process graph and associations of open nodes

as $n \in ON_{pre}(f)$ ($n \in ON_{post}(f)$). A node is a closed node if it is not an open node.

For example, in *graph 6* of Fig. 3, node d is a post-open node for split feature $b \rightarrow \{c, d, e, f, g\}$, and node d is a both pre-open and post-open node for join feature $\{c, d\} \rightarrow h$. When generating a synthetic graph, given $b \rightarrow \{c, d, e, f, g\}$ is already in the synthetic graph, node d can be used to extend the graph with another feature that has a pre-open node. More details about extending a synthetic graph are given in Sect. 4.

For a split or join feature, a subset of open nodes can be used to extend features, because split and join features have more than two endpoints. The goal of extending the open nodes of a split or a join feature is to support a structure where the branches of a split are eventually joined, either immediately or eventually. For example, in Fig. 3, after the split node b , the branches of node c and node d are joined immediately at node h , while the branches of nodes e and f are joined eventually at node j . To support this type of structure, the split and join association of open nodes is defined in Definition 10.

Definition 10 (*Split (join) association of open nodes*) Let g be a business process graph and f be a split feature. Let the node $n \in N_f$ be the split node, and let $N' = ON_{post}(f)$ be the post-open node set for f . Let $N'' \subseteq N'$ ($|N''| > 1$) be a subset of the post-open nodes. If there exists a node $n_2 \in N_g$, such that for each $n_1 \in N''$, $(n_1, n_2) \in E_g$, we say that f has an immediate join, denoted as $\zeta(N'') = Im$; otherwise, if there exists a node $n_2 \in N_g$, such that for each $n_1 \in N''$, there exists a sequence feature in g , $n_1 \rightarrow \dots \rightarrow n_2$, we say that f has an eventual join, denoted as $\zeta(N'') = Ev$. Im and Ev are the types of a split (join) association of open nodes. The join association of open nodes is defined similarly.

For example, the right side of Fig. 3 shows some of the split and join features of *graph 6*. The split feature $b \rightarrow \{c, d, e, f, g\}$ has an immediate join $\{c, d\}$ and an eventual join $\{e, f\}$, while the join feature $\{c, d\} \rightarrow h$ has an immediate split $\{c, d\}$ and the join feature $\{d, i\} \rightarrow j$ has

an eventual split $\{d, i\}$. For an immediate join (split), a join (split) feature is inserted to extend the process graph; for an eventual join (split), a join (split) feature and several sequence features are inserted to extend the process graph.

We now know that a feature can be inserted into a process graph if they both have open nodes or split (join) associations of open nodes. However, the definitions of a feature and a process graph do not contain items to indicate which nodes are open nodes or split (join) associations of open nodes. Therefore, we define a component as a specific type of feature that also records information about the open nodes of that feature. We need this information, because in next section components are composed to form a (partial) synthetic process graphs, and open nodes or split (join) associations of open nodes indicate how to compose components. A component is defined as follows.

Definition 11 (*Component*) Let g be a business process graph, and let f be a local maximum feature of g . The component of f , denoted as $c = Comp(f, g)$, is a tuple $(N, E, \lambda, preON, postON, \zeta)$, in which:

- $N = N_f$ is the set of nodes.
- $E \subseteq N \times N$ is the set of edges, where $E = E_f$.
- λ is a function that maps each node in N to an empty label.
- $preON = \{n \in N_f | (\bullet n - N_f) \neq \emptyset\}$ is the pre-open node set.
- $postON = \{n \in N_f | (n \bullet - N_f) \neq \emptyset\}$ is the post-open node set.
- ζ is a function that maps a subset of the pre-open (post-open) node set to an immediate or eventual join (split), as defined in Definition 10.

The size and type of a component are the same as the size and type of the feature it derives from, i.e., $Size(s) = Size(f)$ and $Type(s) = Type(f)$.

For example, $(\{a, b, g\}, \{(a, b), (b, g)\}, \lambda, \emptyset, \{b\}, \zeta)$ is a sequence component of *graph 6* (labels are used to identify

nodes here). Definition 12 presents how to extract all components from a process graph or a collection of process graphs.

Definition 12 (*Component Set*) Let $\mathcal{D}$ be a collection of process graphs. The component set of $\mathcal{D}$ consists of the components of all local maximums of process graphs in $\mathcal{D}$. Formally, $C = \{Comp(f, g) | g \in \mathcal{D} \wedge f \in \Gamma(g) \cap LMT(g)\}$.

We say two components are equivalent if there is a mapping between two components, as defined in Definition 13.

Definition 13 (*Component equivalence*) Let $c_1 = (N_1, E_1, \lambda, preON_1, postON_1, \zeta)$ and $c_2 = (N_2, E_2, \lambda, preON_2, postON_2, \zeta)$ be two components. Components c_1 and c_2 are equivalent, denoted as $c_1 = c_2$, if and only if there exists a one-to-one mapping $M : N_1 \rightarrow N_2$, such that

- $\forall n \in preON_1: M(n) \in preON_2; \forall n \in postON_1: M(n) \in postON_2; \forall n \in (N_1 - preON_1 - postON_1): M(n) \in (N_2 - preON_2 - postON_2);$
- $\forall (n, m) \in E_1: (M(n), M(m)) \in E_2;$
- if s_1 and s_2 are split (join) components, $\forall sN_1 \in postON_1(preON_1): \zeta_1(sN_1) = \zeta_2(\{M(n_1) | n_1 \in sN_1\})$.

For example, in Fig. 3, components $\{c, d\} \rightarrow h$ and $\{d, i\} \rightarrow j$ are not equivalent, because $\zeta(\{c, d\}) \neq \zeta(\{d, i\})$. Based on the component equivalence, the frequency and probability of component occurrence are defined as follows.

Definition 14 (*Frequency/probability of component occurrence*) Let $\mathcal{D}$ be a collection of process graphs and C be the component set of $\mathcal{D}$. Let $c \in C$ be a component. The frequency of the occurrence of c is the number of local maximum features in $\mathcal{D}$ that have a component that is equivalent with c , denoted as $FCO(c, C) = |\{f \in \Gamma(g) | g \in \mathcal{D} \wedge f \in LMF(g) \wedge Comp(f, g) = c\}|$. The probability of its occurrence is the fraction of the frequency of its occurrence and the frequency of the occurrence of all components, denoted as $PCO(c, C) = \frac{FCO(c, C)}{\sum_{c_1 \in C} FCO(c_1, C)}$.

For example, in Fig. 3, the frequency of component $\{c, d\} \rightarrow h$ is 1 and the probability of its occurrence is 1/7.

3.3 Node- and edge-type properties

In BPMN 2.0, nodes and edges can be of different types. For example, a node can be a user task, an intermediate catch event, or an exclusive gateway. To generate models with similar properties for node and edge types in a collection, we consider three kinds of properties regarding node and edge types: the occurrence of a node type; the co-occurrence of node types between two connected nodes; and the co-occurrence of an edge type with its input and output node types. The occurrence of a node type concerns the frequency

and probability of a node type appearing in a collection of models; the co-occurrence of node types concerns the frequency and probability of two node types co-occurring for two connected nodes; the co-occurrence of an edge type concerns the frequency and probability of an edge type co-occurring with the two node types of the edge's input and output nodes.

To assign a type to a node, it is necessary to know the probability of node-type occurrence. The type of a node depends on the structural properties of that node, in particular on the number of incoming and outgoing edges of that node. For example, a node that has no incoming edges is more likely to be a start event, while a node that has multiple incoming or outgoing edges is more likely to be a gateway. Note, however, that these are probabilities; a node with multiple incoming or outgoing edges can also be a task. Consequently, the frequency and probability of node types depend on the number of edges of the nodes, and we define them as follows.

Definition 15 (*Frequency/probability of node-type occurrence*) Let N be a set of nodes. The indegree type of a node is the characterization of the number of incoming edges of that node as either 0, 1, or many, such that given a node $n \in N$,

$$indegreetype(n) = \begin{cases} 0 & \text{if } |\bullet n| = 0 \\ 1 & \text{if } |\bullet n| = 1 \\ \text{many} & \text{otherwise} \end{cases}$$

The outdegreetype is defined analogously.

The frequency of the occurrence of a node type t for nodes with an indegree type i and an outdegree type o is:

$$FNT O(t, i, o) = |\{n \in N | \mu(n) = t, \text{ indegreetype}(n) = i, \text{ outdegreetype}(n) = o\}|.$$

The probability that a node n has a particular type t is

$$PNT O(n, t) = \frac{FNT O(t, \text{ indegreetype}(n), \text{ outdegreetype}(n))}{\sum_{t \in \mathcal{T}_N} FNT O(t, \text{ indegreetype}(n), \text{ outdegreetype}(n))}$$

For example, in the process graph of Fig. 2, there are 11 nodes in total. The frequency of user task is 2, and both of them have one input and one output edges, i.e., $FNT O(\text{"userTask"}, 1, 1) = 2$. There are 5 elements that have one input and one output edges in this process model; therefore, its occurrence probability is $2/5 = 40\%$;

It is also necessary to know which two node types can co-occur for two connected nodes of a process model. The frequency and probability of node-type co-occurrence is defined in Definitions 16 and 17, respectively.

Definition 16 (*Frequency of node-type co-occurrence*) Let $\mathcal{D}$ be a collection of process graphs with disjoint sets of nodes, $\mathcal{N}$ be the node set of $\mathcal{D}$, and $\mathcal{T}_{\mathcal{N}}$ be the node-type set of $\mathcal{N}$.

- Frequency of pre-node-type co-occurrence ($FNTCO_{pre}$): Given two nodes n and n_1 , the frequency of the pre-node-type co-occurrence of $t_1 = \mu(n_1)$ and $t = \mu(n)$ is the number of process fragments (a sequence of two nodes) that satisfy that there exists a process graph $g \in \mathcal{D}$, in which n_1 is in the pre-set of n . Formally, $FNTCO_{pre}(t, t_1) = |\{(n, n_1) \in \mathcal{N} \times \mathcal{N} | g \in \mathcal{D} \wedge n, n_1 \in N_g \wedge n_1 \in \bullet n \wedge t_1 = \mu(n_1) \wedge t = \mu(n)\}|$. We say that node type t_1 pre-co-occurs with node type t in a pre-set node, if $FNTCO_{pre}(t, t_1) > 0$.
- Frequency of post-node-type co-occurrence ($FNTCO_{post}$): Given two nodes n and n_1 , the frequency of the post-node-type co-occurrence of $t_1 = \mu(n_1)$ and $t = \mu(n)$ is the number of process fragments (a sequence of two nodes) that satisfy that there exists a process graph $g \in \mathcal{D}$, in which n_1 is in the post-set of n . Formally, $FNTCO_{post}(t, t_1) = |\{(n, n_1) \in \mathcal{N} \times \mathcal{N} | g \in \mathcal{D} \wedge n, n_1 \in N_g \wedge n_1 \in n \bullet \wedge t_1 = \mu(n_1) \wedge t = \mu(n)\}|$. We say that node type t_1 post-co-occurs with node-type t in a post-set node, if $FNTCO_{post}(t, t_1) > 0$.

Definition 17 (*Probabilities of node-type co-occurrences*) Let $\mathcal{N}$ be the node set of a collection of process graphs with disjoint sets of nodes, and $\mathcal{T}_{\mathcal{N}}$ be the node-type set of $\mathcal{N}$. The probability of node-type co-occurrence is defined as follows.

- Probability of pre-node-type co-occurrence ($PNTCO_{pre}$): Given two nodes $n, n_1 \in \mathcal{N}$, with type $t = \mu(n)$ and $t_1 = \mu(n_1)$, the probability of the co-occurrence of t_1 and t is the frequency of t_1 co-occurring with t in a pre-set node, divided by the frequency of all node types that co-occur with t in a pre-set node. Formally,

$$PNTCO_{pre}(t, t_1) = \frac{FNTCO_{pre}(t, t_1)}{\sum_{t_2 \in \mathcal{T}_{\mathcal{N}}} FNTCO_{pre}(t, t_2)}. \quad (4)$$

- Probability of post-node-type co-occurrence ($PNTCO_{post}$): Given two nodes $n, n_1 \in \mathcal{N}$, with type $t = \mu(n)$ and $t_1 = \mu(n_1)$, the probability of the co-occurrence of t_1 and t is the frequency of t_1 co-occurring with t in a post-set node, divided by the frequency of all node types that co-occur with t in a post-set node. Formally,

$$PNTCO_{post}(t, t_1) = \frac{FNTCO_{post}(t, t_1)}{\sum_{t_2 \in \mathcal{T}_{\mathcal{N}}} FNTCO_{post}(t, t_2)}. \quad (5)$$

For example, in the process graph of Fig. 2, considering the node-type ‘parallel gateway’, there is 1 node with the

type ‘exclusive gateway’ and there are 2 nodes with the type ‘task’ in their pre-sets. There is 1 node with the type ‘end event’ and there are 2 nodes with the type ‘task’ in their post-sets. The frequency of task pre-co-occurrence with ‘parallel gateway’ is 2, and its co-occurrence probability is $2/3 = 66.7\%$; the frequency of ‘end event’ pre-co-occurrence with ‘parallel gateway’ is 1, and its co-occurrence probability is $1/3 = 33.3\%$.

To assign an edge type to an edge, the types of the edge’s input and output nodes are considered. This is because the edge type is highly correlated with the type of its input and output nodes. For example, in BPMN, two flow objects are connected by a sequence flow, while a flow object and a message object are connected by a message flow. The frequency and probability of an edge type occurring with the types of input and output nodes of the edge are defined in Definitions 18 and 19, respectively.

Definition 18 (*Frequency of edge-node-type co-occurrence*) Let $\mathcal{D}$ be a collection of process graphs with disjoint sets of nodes, $\mathcal{N}$ and $\mathcal{E}$ be the node and edge sets of $\mathcal{D}$, respectively, $\mathcal{T}_{\mathcal{N}}$ be the node-type set of $\mathcal{N}$, and $\mathcal{T}_{\mathcal{E}}$ be the edge-type set of $\mathcal{E}$.

Frequency of edge-node-type co-occurrence ($FETCO$): Given two node types t, t_1 , and an edge type t_e , the frequency of the co-occurrence of t_e with node types t and t_1 is the number of process fragments (a sequence of two nodes) that satisfy that there exists a process graph $g \in \mathcal{D}$, in which n_1 is in the post-set of n , and the types of n, n_1 , and t_e are t, t_1 , and t_e , respectively. Formally, $FETCO(t_e, t, t_1) = |\{(n, n_1) \in \mathcal{E} | g \in \mathcal{D} \wedge n, n_1 \in N_g \wedge n_1 \in n \bullet \wedge t_1 = \mu(n_1) \wedge t = \mu(n) \wedge t_e = v((n, n_1))\}|$. We say that node type t_1 co-occurs with node type t in a pre-set node, if $FETCO_{pre}(t_e, t, t_1) > 0$.

Definition 19 (*Probability of edge-node-type co-occurrence*) Probability of edge-node-type co-occurrences ($PENTCO$): Given two node types t, t_1 , and an edge type t_e , the probability of the co-occurrence of t_e and t and t_1 is the frequency with which t_e co-occurs with t and t_1 divided by the frequency of all edge types co-occurring with t and t_1 . Formally,

$$PENTCO(t_e, t, t_1) = \frac{FETCO(t_e, t, t_1)}{\sum_{t'_e \in \mathcal{T}_{\mathcal{E}}} FETCO(t'_e, t, t_1)}. \quad (6)$$

Continuing with the example of the process graph of Fig. 2. Considering the sequence flows between parallel gateways and user tasks, the frequency of their Edge-Node-Type Co-Occurrences is 2 and the probability is 100%.

4 Generating synthetic process model collections

This section presents the algorithm to generate synthetic process models based on the properties defined in the pre-

vious section. The algorithm consists of two steps. first, it generates node labels, and then, it generates a synthetic graph by inserting components into the graph and labeling component nodes.

Synthetic labels can be generated based on the probability of word occurrence and the probability of word co-occurrence, which is defined in Definition 20.

Definition 20 (*Synthetic label*) Let $\mathcal{W}$ be the word set of a collection of process graphs, and $\omega(l)$ be the function that maps a label l to the set of words that appear in l .

A synthetic label of size s consists of a word, $w \in \mathcal{W}$, and a set of $s - 1$ words, $W = \{w_1, w_2, \dots, w_{s-1}\} \subset \mathcal{W}$ ($w \notin W$), which co-occur with w .

The probability of the size s is $\frac{|\{n \in \mathcal{N} | \omega(\lambda(n)) = s\}|}{|\mathcal{N}|}$. The probability of selecting a word w is $PWO(w)$, according to Definition 3. The probability of selecting a word $w_i \in W$ is $PWCO(w, w_i)$, according to Definition 4.

For example, “receive goods online” and “order application” can be synthetic labels for the process graphs in Fig. 1.

A synthetic node is labeled by selecting a synthetic label from a set of synthetic labels. The selection consists of two steps. First, a word is selected based on the probabilities of pre-word and post-word co-occurrence. Second, a synthetic label is selected from the subset of synthetic labels that contain the selected word. The probability of selecting a label is defined as follows.

Definition 21 (*Probability of label selection*) Let $\mathcal{SL}$ be a set of synthetic labels generated according to Definition 20, let sl be a synthetic label, $sl \in \mathcal{SL}$, and let $\mathcal{W} = \{w | w \in \omega(sl) \wedge sl \in \mathcal{SL}\}$ be the word set of $\mathcal{SL}$. Let c be a component, and let n be a node of c , $n \in N_c$.

The probability of selecting a word w is the frequency of words in the pre-set/post-set of n that co-occur with w in a pre-set/post-set node label, divided by the frequency of words in the pre-set/post-set of n that co-occur with any word in a pre-set/post-set node label. If the denominator is 0, the probability of selecting w is the probability of the occurrence of w .

$$PLS(w) = \begin{cases} \frac{\sum_{w' \in W_{post}} FWCO_{pre}(w', w) + \sum_{w' \in W_{pre}} FWCO_{post}(w', w)}{\sum_{w'' \in \mathcal{W}} (\sum_{w' \in W_{post}} FWCO_{pre}(w', w'') + \sum_{w' \in W_{pre}} FWCO_{post}(w', w''))}, & \text{if } \sum_{w'' \in \mathcal{W}} (\sum_{w' \in W_{post}} FWCO_{pre}(w', w'') + \sum_{w' \in W_{pre}} FWCO_{post}(w', w'')) \neq 0; \\ PWO(w), & \text{otherwise.} \end{cases}$$

where $W_{post} = \{w | \forall n_1 \in n \bullet \wedge w \in \omega(\lambda(n_1))\}$ and $W_{pre} = \{w | \forall n_1 \in \bullet n \wedge w \in \omega(\lambda(n_1))\}$.

The probability of selecting a label sl from $\mathcal{SL}$ based on w is 1, divided by the number of labels that contains the word w , i.e., $P_l(w, sl) = \frac{1}{|\{sl_1 \in \mathcal{SL} | w \in \omega(sl_1)\}|}$.

Overall, the probability of selecting a synthetic label sl from $\mathcal{SL}$ is $P(sl) = \sum_{w \in \omega(sl)} (P_w(w) \times P_l(w, sl))$.

To generate a synthetic graph, the size of the synthetic graph is required, of which the probability is defined as follows.

Definition 22 (*Probability of graph size*) Let $\mathcal{D}$ be a collection of process graphs. The probability of a graph g of size s in $\mathcal{D}$, $PGS(s, \mathcal{D})$, is that the number of graphs of size s in $\mathcal{D}$ is divided by the number of all graphs in $\mathcal{D}$. Formally, $PGS(s, \mathcal{D}) = \frac{|\{g \in \mathcal{D} | |E_g| = s\}|}{|\mathcal{D}|}$.

For example, of the five process graphs in Fig. 1, three are of size 1, one is of size 2, and one is of size 4. Therefore, the probability of size 1 is 0.6.

Algorithm 1 presents the algorithm for generating synthetic process graphs. First, a set of synthetic labels is generated, according to Definition 20. These labels are used to label nodes in components later on (line 2). Second, a set of synthetic process graphs is generated. Each graph is generated as follows. Initially, a synthetic process graph sg contains one node n that is a both pre-open and post-open node ($n \in preON \cap postON$). The label of n is empty, and the synthetic process graph does not contain any edge (line 7). Then, components are inserted to extend the graph (lines 9–25).

To insert a component into a synthetic graph, an open node is selected at random. If the component is a loop feature, the open node in the loop node set is recorded. If the open node is not in a join (split) association, a component is selected that has an open node that can be merged with the open node in the synthetic graph as explained in Algorithm 2. If the open node is in a join (split) association, one of those join (split) associations is selected at random. Then, a component is selected, which has a join (split) association with the same number of open nodes and the same type (line 23). The probability of the selection is the probability $PCO(c, C_1)$ as explained in Definition 14. Finally, the open nodes in the join (split) associations are merged as explained in Algorithm 3.

Algorithm 2 presents the steps of merging an open node of a synthetic graph and an open node of a component. First, (open) nodes in the component are inserted into the (open) node set of the synthetic graph, such that the open nodes that are to be merged are not open anymore. Second, edges that are connected to the open node in the component are connected to the open node in the synthetic graph. Other edges in the component are inserted into the edge set of the synthetic graph. Third, nodes in the component—except for the open nodes—are labeled according to Definition 21.

Algorithm 3 presents the steps of merging a join (split) association of open nodes of a synthetic graph and a split (join) association of open nodes of a component. If the associations are immediate, the pair of nodes are merged

Algorithm 1: Synthetic Process Graph Generation

input : a collection of process graphs: $\mathcal{D}$, an integer: $size_c$
output: a collection of synthetic process graphs $\mathcal{SD}$

```

1 begin
2    $\mathcal{SL}$  is the set of generated synthetic labels (Definition 20);
3    $\mathcal{SD} \leftarrow \emptyset$ ;
4    $LN \leftarrow \emptyset$ ; //Set of loop nodes.
5    $C$  is the component set of  $\mathcal{D}$  (Definition 12);
6   while  $|\mathcal{SD}| < size_c$  do
7      $sg = \{\{sn\}, \emptyset, \emptyset, \{sn\}, \{sn\}, \emptyset\}$ ; //sn is a newly created
8     node.
9     select  $size_g \in \{|E_g| | g \in \mathcal{D}\}$  with probability
10     $PGS(size_g, \mathcal{D})$  (Definition 22);
11    while  $|E_{sg}| < size_g \wedge (preON_{sg} \cup postON_{sg}) \neq \emptyset$  do
12      randomly select  $n \in preON_{sg} \cup postON_{sg}$ ;
13      if  $n \in postON_{sg}$  then
14        if  $\nexists N \subseteq N_{sg} : n \in N \wedge \zeta_{sg}(N) \in \{Im, Ev\}$  then
15           $C_1 \leftarrow \{c \in C | preON_c \neq \emptyset\}$ ;
16          select  $c \in C_1$  with probability  $PCO(c, C_1)$ 
17          (Definition 14);
18          if  $c$  is a loop feature then
19             $LN \leftarrow LN \cup \{sg\}$ ;
20          else
21            randomly select  $n_1 \in preON_c$ ;
22             $sg \leftarrow \text{mergeNode}(sg, c, n, n_1, \mathcal{SL})$ ;
23            //Algorithm 2.
24          else
25            randomly select  $AN \in \{N \subseteq N_{sg} | n \in$ 
26             $N \wedge \zeta_{sg}(N) \in \{Im, Ev\}\}$ ;
27             $C_1 \leftarrow \{c \in C | Type(c) = join \wedge AN \subseteq$ 
28             $preON_c \wedge |AN| = |AN_1| \wedge \zeta_{sg}(AN) =$ 
29             $\zeta_c(AN_1)\}$ ;
30            select  $c \in C_1$  with probability  $PCO(c, C_1)$ 
31            (Definition 14);
32             $sg \leftarrow \text{mergeAsso}$ 
33             $(sg, c, AN, AN_1, \mathcal{SL})$ ; //Algorithm 3.
34          if  $n \in preON_{sg}$  then //similar to lines 12–27. ;
35          if  $|LN| > 1$  then  $sg \leftarrow \text{constructLoop}$ 
36           $(sg, LN)$ ; //Algorithm 4.;
37           $sg \leftarrow \text{assignType}(sg)$ ; //Algorithm 5.
38           $\mathcal{SD} \leftarrow \mathcal{SD} \cup \{sg\}$ ;
39 return  $\mathcal{SD}$ ;

```

(Algorithm 2). If the associations are eventual, then a sequence component is selected to connect each open node in the synthetic graph to an open node in the component. The connection is made by merging the pair of open nodes with two open nodes in the sequence component (lines 7–11). Subsequently, the join (split) associations of the open nodes are updated, and the merged associations are deleted. If there are other split and join associations in the components, then these associations are recorded in the synthetic graph (lines 12–15).

Algorithm 4 shows how to construct a loop in a synthetic process graph. This is done by randomly selecting two nodes from the given loop node set and adding edges between them if there is no path between them.

Algorithm 2: Merge a Pair of Open Nodes

input : a synthetic process graph: sg , a component: c , an open node of sg : n_{sg} , an open node of c : n_c , a synthetic label set: $\mathcal{SL}$
output: a synthetic process graphs: sg

```

1 begin
2    $N_{sg} \leftarrow N_{sg} \cup (N_c - \{n_c\})$ ;
3   if  $n_{sg} \in preON_{sg} \wedge n_c \in postON_c$  then
4      $preON_{sg} \leftarrow (preON_{sg} - \{n_{sg}\}) \cup preON_c$ ;
5      $postON_{sg} \leftarrow postON_{sg} \cup (postON_c - \{n_c\})$ ;
6   else if  $n_{sg} \in postON_{sg} \wedge n_c \in preON_c$  then
7      $preON_{sg} \leftarrow preON_{sg} \cup (preON_c - \{n_c\})$ ;
8      $postON_{sg} \leftarrow (postON_{sg} - \{n_{sg}\}) \cup postON_c$ ;
9    $E_{sg} \leftarrow E_{sg} \cup \{(n, n_3) | n_3 \in n_1 \bullet\} \cup \{(n_3, n) | n_3 \in \bullet n_1\}$ 
10    $\cup (E_c - \{(n_1, n_3) | n_3 \in n_1 \bullet\} - \{(n_3, n_1) | n_3 \in \bullet n_1\})$ ;
11   foreach  $n \in (postON_c - \{n_c\})$  do
12     select a label  $sl$  from  $\mathcal{SL}$  with probability
13      $P(sl)$  (Definition 21).
14      $\lambda_{sg} \leftarrow \lambda_{sg} \cup \{(n, sl)\}$ ;
15 return  $sg$ ;

```

Algorithm 3: Merge Join and Split Associations of Open Nodes

input : a synthetic process graphs: sg , a component: c , a join (split) association of sg : AN_{sg} , a split (join) association of c : AN_c , a synthetic label set: $\mathcal{SL}$
output: a synthetic process graphs: sg

```

1 begin
2   foreach  $n_{sg} \in AN_{sg}$  do
3     randomly select  $n_c$  from  $AN_c$ ;
4      $AN_c \leftarrow AN_c - \{n_c\}$ ;
5     if  $\zeta_{sg}(AN_{sg}) = Im$  then
6        $sg \leftarrow \text{mergeNode}(sg, c, n_{sg}, n_c, \mathcal{SL})$ ; //Algorithm 2.
7     else if  $\zeta_{sg}(AN_{sg}) = Ev$  then
8        $C_1 \leftarrow \{c_1 \in C | Type(c_1) = sequence \wedge (\exists (n_1 \in$ 
9        $preON_{c_1} \wedge n_2 \in postON_{c_1}), n_1 \rightarrow \dots \rightarrow$ 
10       $n_2 \text{ is a subgraph of } c_1)\}$ ;
11      select  $c_1 \in C_1$  with probability  $FCO(c_1, C_1)$ 
12      (Definition 14);
13       $sg \leftarrow \text{mergeNode}(sg, c_1, n_{sg}, n_1, \mathcal{SL})$ ; //Algorithm
14      2.
15       $sg \leftarrow \text{mergeNode}(sg, c, n_2, n_c, \mathcal{SL})$ ; //Algorithm 2.
16   Delete  $AN_{sg}$  from the domain of  $\zeta_{sg}$ ;
17   Delete  $AN_c$  from the domain of  $\zeta_c$ ;
18   foreach  $AN_1 \in \{N_1 \subseteq N_c | \zeta_c(N_1) \in \{Im, Ev\}\}$  do
19      $\zeta_{sg}(AN_1) \leftarrow \zeta_c(AN_1)$ ;
20 return  $sg$ ;

```

Algorithm 5 presents the steps for assigning a type to each node/edge. If a node is a gateway, then its type is the same as its label (lines 2–4), because gateways are labeled with their type during structure and label generation. Gateway types are assigned at that stage to maintain the co-occurring of gateways. For example, an XOR split is likely to be closed at some stage by an XOR join. If a node is a start or an end node, its type is assigned based on the probability of node-

Algorithm 4: Construct Loop in a Synthetic Process Graph

input : a synthetic process graphs: sg , a set of (loop) nodes: LN
output: a synthetic process graphs: sg

```

1 begin
2   randomly select two different nodes  $n_i, n_j$  from  $LN$ ;
3   if there exists not a path from  $n_i$  to  $n_j$  then
4      $E_{sg} \leftarrow E_{sg} \cup \{(n_i, n_j)\}$ 
5   if there exists not a path from  $n_j$  to  $n_i$  then
6      $E_{sg} \leftarrow E_{sg} \cup \{(n_j, n_i)\}$ 
7   return  $sg$ ;

```

type occurrence defined in Definition 15 (lines 5–9). This is because start and end nodes are more likely to be start and end events and this property can be captured by Definition 15, but not by Definition 17, which is used later assigning types for other nodes. After that, the graph is traversed and the type of a visited node is assigned based on the probability of the type co-occurrence with the type of the connected node that was just traversed, as defined in Definition 17 (lines 10–23). Finally, each edge is assigned a type based on the probability of the type co-occurrence with the types of the edge's input and output nodes, as defined in Definition 19 (lines 24–28). After assigning the types to nodes and edges, the graph is returned.

We implemented our algorithm in Java and made it available as an open source project.³

5 Evaluation

This section presents the evaluation of the algorithms proposed in this paper. In the evaluation, we aim to verify that a collection of models that is generated by our algorithm has properties that are similar to the properties of the collection of models from which it is generated. Two evaluations were done: one with 604 SAP reference models in the EPC notation and one with 22 models in the BPMN 2.0 notation. We use the first evaluation to determine to which extent our algorithm can generate a model collection with properties that are similar to the original collection. We use the second evaluation to investigate to which extent these properties are also preserved for BPMN 2.0 model generation, although this collection is not large enough to do this statistically. While only two collections are used as input to generate synthetic collections in experiments, we argue that these two collections are representative for other collections in terms of their properties. Therefore, the results of the evaluation should carry over to other collections from practice as well. However, in

Algorithm 5: Assign Types to Nodes and Edges

input : a process graph (without node-type values): sg
output: a process graph (with assigned node-type values): sg

```

1 begin
2   //Gateways
3   foreach  $n \in \mathcal{N}_{sg} \wedge (|\bullet n| > 1 \vee |n \bullet| > 1)$  do
4     if  $\mu_{sg}(n) = null$  then  $\mu_{sg} \leftarrow \mu_{sg} \cup \{(n, \lambda_{sg}(n))\}$ ;
5   //Start and end nodes
6   foreach  $n \in \mathcal{N}_{sg} \wedge (|\bullet n| = 1 \vee |n \bullet| = 1)$  do
7     if  $\mu_{sg}(n) = null$  then
8       select a node type  $t$  from  $\mathcal{T}_{\mathcal{N}}$  with probability
         $PNT O(t, indegreetype(n), outdegreetype(n))$ 
        (Definition 15).
7        $\mu_{sg} \leftarrow \mu_{sg} \cup \{(n, t)\}$ ;
9   //Other nodes
10   $N \leftarrow \{n \in \mathcal{N}_{sg} | (|\bullet n| > 1 \vee |n \bullet| > 1)\}$ ;
11  while  $N \neq \emptyset$  do
12    Randomly select  $n \in N$ ;  $N \leftarrow N - \{n\}$ ;
13    foreach  $n_{pre} \in \bullet n$  do
14      if  $\mu_{sg}(n_{pre}) = null$  then
15        select a node type  $t$  from  $\mathcal{T}_{\mathcal{N}}$  with probability
16         $PNTCO_{pre}(\mu_{sg}(n), t)$  (Definition 17).
17         $\mu_{sg} \leftarrow \mu_{sg} \cup \{(n_{pre}, t)\}$ ;
18         $N \leftarrow N \cup \{n_{pre}\}$ ;
19    foreach  $n_{post} \in n \bullet$  do
20      if  $\mu_{sg}(n_{post}) = null$  then
21        select a node type  $t$  from  $\mathcal{T}_{\mathcal{N}}$  with probability
22         $PNTCO_{post}(\mu_{sg}(n), t)$  (Definition 17).
23         $\mu_{sg} \leftarrow \mu_{sg} \cup \{(n_{post}, t)\}$ ;
24         $N \leftarrow N \cup \{n_{post}\}$ ;
25  //Edges
26  foreach  $e = (n, n_1) \in \mathcal{E}_{sg}$  do
27    if  $v_{sg}(e) = null$  then
28      select an edge type  $t$  from  $\mathcal{T}_{\mathcal{E}}$  with probability
29       $PENTCO(t, \mu_{sg}(n), \mu_{sg}(n_1))$  (Definition 19).
29     $v_{sg} \leftarrow v_{sg} \cup \{(e, t)\}$ ;
30  return  $sg$ ;

```

future work, evaluations with additional collections would help to better establish the generalizability of the results.

For the first evaluation, we ran the algorithm 100 times with 604 SAP reference models as input. In total, 60,400 synthetic BPMN models were generated in 15:50 min. Figure 4 shows the distributions of the sizes of both the original collection of models and the generated collection of models. The distributions appear to be very similar. Table 1 shows a more detailed comparison of the properties of the SAP reference models and the synthetic models, including their descriptive statistics. The properties include the average number of words in a label, the average number of occurrences of a label, the average number of nodes and edges in a process model, and the average number of local maximum split, join, loop, and sequence features in a model. In the last row, the table shows whether the differences between the model collections are significant for a particular property and, if so, at

³ <https://github.com/rmdijkman/modelsynthesis>.

Fig. 4 Model size distributions **Table 1** Properties of models

	Words	Label occ.	Nodes	Edges	Splits	Joins	Sequences	Loops
SAP								
Mean	2.71	1.92	20.71	20.77	2.63	2.58	2.71	0.05
SD	0.62	0.56	18.70	20.80	3.41	3.25	2.20	0.23
Skewness	0.96	1.35	2.30	2.43	2.87	2.45	1.65	3.91
Kurtosis	1.52	2.83	6.79	7.48	12.26	7.49	3.74	13.31
Synthetic								
Mean	3.75	1.29	19.27	20.89	2.34	2.19	1.77	0.06
SD	0.66	0.18	18.07	21.82	2.79	2.65	1.78	0.23
Skewness	0.98	1.18	2.18	2.34	2.59	2.53	1.99	3.88
Kurtosis	4.30	5.75	6.18	7.17	9.16	8.89	5.75	13.06
Difference								
Significance	$p < 0.01$	$p < 0.01$			$p < 0.05$	$p < 0.01$	$p < 0.01$	

which level. We used a t test to determine this number. The two datasets did not have equal variance. Therefore, we used the t test alternative for datasets with unequal variance. While the datasets were not normally distributed, a t test could be applied because the number of datapoints in the datasets (604 and 60,400) was large enough [3]. The table shows, in line with the findings in Fig. 4, that there is no significant difference between the sizes of the models, neither in terms of the number of nodes, nor in terms of the number of edges. Also, there is no significant difference between the number of occurrences of loops in the models. There do exist significant differences with respect to the other properties. The main differences exist with respect to the average number of words per label (2.71 for the original models and 3.75 for the generated models) and with respect to the average number of sequences in a model (2.71 for the originals model and 1.77 for the generated models).

It is to be expected that (at least some) of the differences will increase when the size of the models increases. Table 2 illustrates this potential effect. This table shows the difference between the model collections in terms of mean absolute error, for selected sizes of models: models with 0–5 nodes, models with 6–10 nodes,... The table shows that

the difference in the number of words per label is constant as the size of the model grows, while the difference in the number of sequences appears to increase as the size of the model grows. This also holds for the difference with respect to the number of joins. Another striking finding is that, while the difference in the number of edges was insignificant between the model collections, the difference does appear to grow.

From this evaluation, we conclude that, while there are significant differences between properties of a generated and those of an original model collections, the synthetic collection also shows substantial similarity to the original collection. Therefore, we claim that the algorithm can be used to generate a much larger model collection that has some properties that are similar to the collection from which it was derived. At the same time, the collection also has different properties, especially with respect to the labels that are generated. Consequently, future work on the generation of synthetic labels is still necessary. In addition, one should be careful when using a synthetic collection for evaluating techniques that do advanced label analysis, while a synthetic collection can be suitable to evaluate techniques that do basic label analysis.

Table 2 Differences between properties

Model size (nodes)	0...5	5...10	10...15	15...20	20...25	25...30	30...35	35...40
Words	1.03	0.86	1.09	1.00	1.01	0.94	1.13	1.16
Label occ.	0.35	0.56	0.50	0.72	0.77	0.81	0.72	0.88
Nodes	0.85	0.22	0.33	0.32	0.11	0.07	0.20	0.14
Edges	0.86	0.31	1.52	1.11	1.93	3.02	2.48	4.15
Splits	0.09	0.12	0.03	0.12	0.29	0.15	0.55	0.13
Joins	0.09	0.07	0.08	0.14	0.09	0.23	0.80	1.45
Sequences	0.47	0.44	0.57	1.12	1.08	0.86	1.29	1.44
Loops	0.01	0.00	0.00	0.02	0.03	0.01	0.04	0.06

Fig. 5 Example of a synthetic BPMN model

For the second evaluation, we ran the algorithm 100 times with 22 BPMN models as input. In total, 2200 synthetic BPMN models were generated in 0:08 min. Figure 5 shows an example of a generated synthetic BPMN Model. This example shows that reasonable process models are generated, although—as expected—the labels of the generated models do not always make sense. The model, for example, contains labels like ‘for level provide support’ next to labels like ‘example problem’.

The example model was generated as follows. The generation started with the task “for level provide support”. On the left side, a sequence of three nodes was inserted. On the right side, a split with two parallel branches that are eventually joined was inserted. Then, a sequence with three nodes and a sequence with two nodes were inserted to connect the branches of the join and the split patterns.

Table 3 shows the properties of the original OMG BPMN models and the synthetic models. The properties that were compared in this evaluation are more detailed than the ones that were compared in the previous evaluation (viz. Table 1). In particular, in this evaluation, we checked whether attributes of tasks and gateways are preserved. Table 3 shows that the properties of the input collection and the output collection are similar. In the bottom part of the table (object and flow types), we can see that all the elements in the BPMN models that are considered in this paper also appear in the synthetic BPMN models in this experiment. Note that this may not be always the case, because elements with low occurrence rates, such as intermediate throw events and event-based gateways in Table 3, may be absent in the synthetic collection.

We also investigated the level of similarity between the original BPMN models and the synthetic BPMN models.

To that end, we took the most similar synthetic BPMN model for each original BPMN model and presented the similarity between the two models (on a zero-to-one scale). The similarity was computed using the technique described in [30]. Table 4 shows the results. As expected, the level of similarity between the original and the synthetic models is low. This can be attributed to the differences between their node labels, which play an important role when computing similarity. The process models do have similar structures, but quite different labels, motivating future work in the area of automatic label generation.

The evaluation shows that the algorithms can indeed be used to generate a much larger BPMN model collection that has properties that are similar to the properties of the model collection from which it is derived. In particular, it captures the occurrence probabilities of different BPMN elements and attributes of those elements. While the labels that are generated by the algorithm serve purposes such as evaluating the computational performance of algorithms for process similarity search, querying, and matching (as shown in, for example, [31] and [33]), our evaluation also shows that more advanced techniques for label matching (such as [28]) require more advanced methods for label generation. This is left as a topic for future work.

6 Related work

This section discusses related work in both the business process model and general graph generation areas. To the best of our knowledge, the following techniques are the only existing techniques for generating synthetic business process models. van Hee et al. [24] propose a technique for gener-

Table 3 Properties of process models

	Original models	Synthetic models
<i>Words and labels</i>		
Words	2.24	2.37
Label occurrences	1.75	2.25
<i>Structures</i>		
Splits	2.27	1.51
Joins	1.55	1.54
Sequences	2.36	2.14
Loops	0.13	0.08
<i>Object and flow types</i>		
Start event	1.55	0.93
End event	2.18	1.02
Intermediate catch event	0.55	0.76
Intermediate throw event	0.09	0.12
Boundary event	0.27	0.46
Task	2.32	2.80
User task	1.41	1.90
Send task	0.82	0.10
Call activity	0.17	0.32
Script task	0.27	0.20
Manual task	1.23	1.39
Service task	0.14	0.06
Receive task	0.18	0.14
Subprocess	0.55	0.76
Event-based gateway	0.09	0.03
Exclusive gateway	1.64	1.58
Parallel gateway	0.73	0.91
Inclusive gateway	0.09	0.05
Overall (event, activity, and gateway)	14.14	13.53
Sequence flow	14.0	13.86

ating synthetic Petri nets based on a given process model collection. Their technique focuses on structure and does not consider activity labels, labeling activities with codes to distinguish them from each other instead. Also, since the technique focuses on generating Petri nets, it only considers two types of nodes (places and transitions), while our technique considers more types of nodes, such as user tasks, manual tasks, event-based gateways, and boundary events. Jin et al. [10] also propose a technique for generating a collection of Petri nets. They do not provide detailed information about the model generation and only basic information about the generated Petri nets (the number of places, transitions, arcs, and labels).

This paper is an extension version of previous work [32]. The contribution to this paper compared to previous work is that more semantic information is extracted from the original collection of models and added to the generated models.

Table 4 Similarity of process models

Original model	Most similar synthetic model	Similarity
1	42	0.34
2	42	0.34
3	163	0.35
4	159	0.53
5	203	0.44
6	157	0.61
7	159	0.53
8	42	0.34
9	98	0.35
10	144	0.34
11	160	0.30
12	177	0.37
13	177	0.37
14	160	0.37
15	42	0.34
16	42	0.44
17	42	0.34
18	42	0.34
19	10	0.35
20	59	0.35
21	40	0.34
22	190	0.35

Process mining techniques can discover Petri nets [23] and BPMN models [5] from event logs. However, process mining has a different goal, focusing on the generation of accurate models from logs rather than synthetic models from other models.

Model synthesis is studied in different areas, including model transformation synthesis [15] and behavioral model synthesis [11]. Of particular interest is graph synthesis, as our technique is a specific form of graph synthesis. In the area of graph synthesis, Kuramochi et al. [14] generate graphs by inserting fragments into a synthetic graph. Chakrabarti et al. [4] provide tools to generate both random and power-law graphs. Akoglu et al. [1] propose a recursive model to generate realistic graphs. More graph generation techniques are summarized in the survey paper by Bader et al. [2]. Our technique is based on Kuramochi et al. [14] and extends their work with business process specific properties. In particular, it considers the proper joining of process paths, label generation, and attributes of events and tasks.

7 Conclusion

This paper presents a technique for generating large collections of synthetic BPMN process models that have similar

properties to real-life process model collections. The properties that are considered include the size of the process models in the collection, the number of common workflow patterns (sequences, splits, joins, and loops) that appear in the collection, words in the labels of tasks and events, co-occurrence of words, and types of nodes and edges (e.g., start events, user tasks, and default flows). The generated collections can be used for the evaluation of algorithms that are developed for large collections of business process models, such as algorithms for process similarity search, process querying, and process matching.

The evaluation shows that the algorithms can indeed be used to generate a much larger model synthetic collection that has properties that are similar to the model collection from which it is derived. At the same time, the synthetic collection also has properties that are different, especially with respect to the labels that are generated. Consequently, one should only use synthetic collections that are generated with our algorithm for purposes for which they are suitable. We claim that, based on our evaluation, synthetic process model collections are suitable for analyzing the computational complexity of many different algorithms, such as algorithms for process similarity search [10,12,30], process querying [22,31], and process matching [18,26,27,33]. However, they should not be used in the qualitative evaluation of algorithms that consider label properties that go beyond simple bags of words [20,28].

The techniques presented in this paper can be improved in several ways in future work. First, the labels generated in this paper are essentially bags of words, constructed from words that appear in the labels of the original models. In addition, our evaluation shows that the labels that are generated by our algorithm still differ significantly from the labels of the process model collection from which they are derived. Therefore, more advanced ways to generate labels can be considered in future work. Second, the algorithm can be improved by considering other techniques for generation. In particular, model generation could be improved by using the refined process structure tree (RPST) [25] and generating a model in terms of the frequency of occurrence of RPST components. This can both help to improve the performance of our algorithm and the matching of splits and joins, although the generation of joins will still be a challenge for rigid RPST components. Third, the technique in this paper mainly focuses on labels, control flows, and node and edge types. However, process models often contain more information, such as resources and data, which are also useful for detecting differences between models. The technique can be extended to cover these other aspects. Fourth, only a subset of BPMN elements are considered by our technique. This subset can still be extended. In the future, we could consider collaborations, message flows, and activity markers, such as the loop and the sequential multi-instance markers. Finally,

properties of the execution semantics of the synthetic BPMN models are not preserved by our technique, although these properties are also important in the evaluation of some algorithms [19]. Therefore, this aspect can be considered in future work.

Acknowledgements The research leading to the paper is supported by the National Science Fund for Young Scholars of China (Project No. 61402301), the National Science Fund of China (Project No. 61472207), the MOE-CMCC research fund (Project No. MCM20123011), the research fund of Capital University of Economics and Business (Project Nos. 00791462722336 and 00791465730165), and the special fund for innovation of Shandong, China (Project No. 2013CXC30001).

References

1. Akoglu, L., Faloutsos, C.: RTG: a recursive realistic graph generator using random typing. *Data Min. Knowl. Discov.* **19**(2), 194–209 (2009)
2. Bader, D.A., Madduri, K.: Gtgraph: A Synthetic Graph Generator Suite. Atlanta (2006). <http://www.cse.psu.edu/~kxm85/software/GTgraph/gen.pdf>
3. Bartlett, M.: The effect of non-normality on the t distribution. Paper presented at the Mathematical Proceedings of the Cambridge Philosophical Society (1935)
4. Chakrabarti, D., Faloutsos, C.: Graph mining: laws, generators, and algorithms. *ACM Comput. Surv. (CSUR)* **38**(1), 2 (2006)
5. Conforti, R., Dumas, M., García-Bañuelos, L., La Rosa, M.: Beyond tasks and gateways: discovering BPMN models with subprocesses, boundary events and activity markers. In: *Business Process Management: 12th International Conference, Lecture Notes in Computer Science*, vol. 8659, pp. 101–117. Haifa, Israel (2014)
6. Curran, T.A., Keller, G.: SAP R/3 Business Blueprint—Business Engineering mit den R/3-Referenzprozessen. Addison-Wesley, Bonn (1999)
7. Dijkman, R.M., Gfeller, B., Küster, J., Völzer, H.: Identifying refactoring opportunities in process model repositories. *Inf. Softw. Technol.* **53**(9), 937–948 (2011)
8. Dumas, M., Van der Aalst, W.M., Ter Hofstede, A.H.: *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley, New York (2005)
9. Gao, X., Chen, Y., Ding, Z., et al.: Process model fragmentization, clustering and merging: an empirical study. In: *BPM Workshops*, Beijing, China (2013)
10. Jin, T., Wang, J., Wu, N., La Rosa, M., ter Hofstede, A.: Efficient and accurate retrieval of business process models through indexing. In: *Proceedings of the 18th CoopIS, Part I, Crete, Greece, LNCS*, vol. 6426, pp. 402–409. Springer, Berlin (2010)
11. Kim, J., Kang, I., Choi, J.-Y., Lee, I., Kang, S.: Formal synthesis of application and platform behaviors of embedded software systems. *Softw. Syst. Model.* **14**(2), 839–859 (2015)
12. Kunze, M., Weidlich, M., Weske, M.: Behavioral similarity—a proper metric. In: *Proceedings of BPM 2011, LNCS*, vol. 6896, pp. 166–181 (2011)
13. Kunze, M., Weidlich, M., Weske, M.: m3—a behavioral similarity metric for business processes. *Services und ihre Komposition* (2011)
14. Kuramochi, M., Karypis, G.: Frequent subgraph discovery. In: *ICDM* (2001)

15. Lano, K., Rahimi, S.K., Poernomo, I., Terrell, J., Zschaler, S.: Correct-by-construction synthesis of model transformations using transformation patterns. *Softw. Syst. Model.* **13**(2), 873–907 (2014)
16. La Rosa, M., van der Reijers, H.A., Aalst, W.M.P., Dijkman, R.M., Mendling, J., Dumas, M., García-Bañuelos, L.: APROMORE: an advanced process model repository. *Expert Syst. Appl.* **38**(6), 7029–7040 (2011)
17. Li, J., Wen, L., Wang, J., Yan, Z.: Process model storage solutions: proposition and evaluation. In: *The First Asia Pacific Conference on Business Process Management* (2013)
18. Liu, K., Yan, Z., Wang, Y., Wen, L., Wang, J.: Efficient syntactic process difference detection using flexible feature matching. In: *The 2nd Asia Pacific Conference on Business Process Management* (2014)
19. Mendling, J.: *Metrics for Process Models: Empirical Foundations of Verification Error Prediction, and Guidelines for Correctness*, volume 6 of *Lecture Notes in Business Information Processing*. Springer, Berlin (2008)
20. Mendling, J., Recker, J., Reijers, H.A.: On the usage of labels and icons in business process modeling. *Int. J. Inf. Syst. Model. Des. (IJISMD)* **1**(2), 40–58 (2010)
21. Object Management Group (OMG): Business process model and notation (BPMN) version 2.0. <http://www.omg.org/spec/BPMN/2.0/>
22. ter Hofstede, A.H.M., Ouyang, C., La Rosa, M., et al.: APQL: a process-model query language. In: *The First Asia Pacific Conference on Business Process Management*, pp. 23–38 (2013)
23. van der Aalst, W.: *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Science & Business Media, Berlin (2011)
24. van Hee, K., La Rosa, M., Liu, Z., et al.: Discovering characteristics of stochastic collections of process models. In: *Business Process Management*, pp. 298–312 (2011)
25. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. *Data Knowl. Eng. (DKE)* **68**(9), 793–818 (2009)
26. Weidlich, M., Weske, M., Mendling, J.: Change propagation in process models using behavioral profiles. In: *SCC'09*, pp. 33–40. IEEE (2009)
27. Weidlich, M., Dijkman, R.M., Mendling, J.: The ICop framework: identification of correspondences between process models. In: *Proceedings of the 22th CAiSE, Hammamet, Tunisia* (2010)
28. Weidlich, M., Sheetrit, E., Branco, M.C., et al.: Matching Business Process Models Using Positional Passage-Based Language Models. *Conceptual Modeling*. Springer, Berlin (2013)
29. Yan, Z., Dijkman, R.M., Grefen, P.W.P.J.: Business process model repositories—framework and survey. *Inf. Softw. Technol.* **54**(4), 380–395 (2012)
30. Yan, Z., Dijkman, R.M., Grefen, P.W.P.J.: Fast business process similarity search. *Distrib. Parallel Databases* **30**(2), 105–144 (2012)
31. Yan, Z., Dijkman, R.M., Grefen, P.W.P.J.: FNet: an index for advanced business process querying. In: *Proceedings of 10th BPM* (2012)
32. Yan, Z., Dijkman, R.M., Grefen, P.W.P.J.: Generating synthetic process model collections with properties of labeled real-life models. In: *The 2nd Asia Pacific Conference on Business Process Management* (2014)
33. Yan, Z., Wang, Y., Wen, L., Wang, J.: Efficient behavioral-difference detection between business process models. In: *Proceedings of the 22nd Conference on CoopIS, LNCS*, vol. 8841 (2014)

Parallel Databases, Information and Software Technology, and International Conference on Business Process Management. Currently, he is leading a national project about process model indexing, and involving in another national project about process mining.

visitor of New York University, Hasso Plattner Institute, IBM Zurich Research Lab, Humboldt-University Berlin, and Queensland University of Technology. Remco has published over 50 papers in scientific journals, conferences, and workshops. His work appeared in *Information Systems*, *Computers in Industry*, and *Transactions on Software Engineering and Methodology*. He has worked as a researcher in a number of research projects, including the SPICE European project, and the SHOPPS, NLI, ArCo, and A-MUSE national projects. Currently, Remco is project leader of the DAIPEx Dinalog project and the GET Service European project.

Zhiqiang Yan is lecturer at Information School of Capital University of Economics and Business, and a (part-time) researcher at School of Software of Tsinghua University. His research interests include Business Process Management and Business Intelligence. In 2012, he received his Ph.D. from Eindhoven University of Technology. He has published over 10 papers in scientific journals, conferences, and workshops. His work appeared in *Distributed and*

Remco Dijkman is associate professor in Information Systems at Eindhoven University of Technology. His research interests include Business Process Management, Business Intelligence, and Service-Oriented Computing. He received his Ph.D. from the University of Twente. Before working at the University of Twente, he combined work as a research assistant at the Open University of the Netherlands with work as a consultant at Ordina. He has been a

Paul Grefen is a full professor in the School of Industrial Engineering at Eindhoven University of Technology since 2003. He chaired the Information Systems subdepartment from 2006 to 2014. He received his Ph.D. in 1992 from the University of Twente and held assistant and associate professor positions in the Computer Science Department. He was a visiting researcher at Stanford University in 1994. He has been involved in various European research

projects as well as various projects within the Netherlands. He is an editor of the International Journal of Cooperative Information Systems. He is an editor and author of the books on the WIDE and Cross-Work projects and has authored books on workflow management and e-business. He is a member of the Executive Board of the European Supply Chain Forum. His current research covers architectural design of business information systems, inter-organizational business process management, and service-oriented business design and support. He teaches at the M.Sc. and Ph.D. levels at TU/e and at the executive level for TIAS business school.