International Journal of Cooperative Information Systems

BUSINESS PROCESS MODEL ABSTRACTION BASED ON SYNTHESIS FROM WELL-STRUCTURED BEHAVIORAL PROFILES

SERGEY SMIRNOV
MATTHIAS WEIDLICH
Hasso Plattner Institute, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany

JAN MENDLING
Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany

There are several motives for creating process models ranging from technical scenarios in workflow automation to business scenarios in which management decisions are taken. As a consequence, companies typically have different process models for the same process, which differ in terms of granularity. In this context, business process model abstraction serves as a technique that takes a process model as an input and derives a high-level model with coarse-grained activities and the corresponding control flow between them. In this way, business process model abstraction reduces the number of models capturing the same business process on different abstraction levels. In this article, we provide a solution to the problem of deriving the control flow of an abstract process model for the case that an arbitrary grouping of activities is permitted. To this end we use behavioral profiles and prove that the soundness of the synthesized process model requires a notion of well-structuredness of the abstract model behavioral profile. Furthermore, we demonstrate that the activities can be grouped according to the data flow of the model in a meaningful way, and that this grouping does not directly coincides with a structural decomposition of the process, which is generally assumed by other abstraction approaches. This finding emphasizes the need for handling arbitrary activity groupings in business process model abstraction.

Keywords: process models; process model management; synthesis; process model abstraction; behavioral profiles.

1. Introduction

Business process management is a general approach to analyze and redesign companies with the overall goal to improve operational excellence and customer satisfaction. Business process models play an important role in this context since they capture knowledge about processes in an explicit way. In large companies business process initiatives often yield several thousand models. There is a strong desire to capture the same information with less models to decrease maintenance effort, in particular, when several models relate to the same process. Such redundancy may be in place when, for instance, there exists a BPEL model capturing the service orchestration, a

detailed conceptual model describing the work steps, and an overview model for senior management. Business process model abstraction (BPMA) provides techniques to derive abstract models from detailed ones by preserving essential properties and leaving out insignificant details. In this way, maintenance can be centered around the most fine-grained model from which the more abstract models are generated.

BPMA is related to different use cases. In particular, a user study with industry has revealed that getting a quick overview of a detailed process is urgently required in practice [40]. Technically, a more high-level process model has to be derived with more coarse-grained activities and their control flow relations. A corresponding BPMA technique has to tackle two questions. First, which activities should be grouped into more coarse-grained ones? Second, what are the control flow relations between them? Most of the existing work makes structural assumptions regarding the first question, such that the second question becomes trivial, cf. [8,17,23,34]. Meanwhile, these restrictions are often not realistic given the requirements in practice [35,48].

This article addresses the business process model abstraction revising and extending our earlier work [42]. The contribution of the article is twofold. First, we develop a technique discovering control flow relations of an abstract model given an unrestricted grouping of activities in the initial model. Our novel approach builds on behavioral profiles, a mechanism capturing behavioral relations between model activities. We develop an approach for the synthesis of a process model from a behavioral profile. The synthesis builds on the newly defined notion of well-structured behavioral profiles. As compared with the abstract model synthesis reported in [42], this article provides a formal synthesis algorithm along with proving soundness of a process generated from a well-structured profile. Second, we design an approach for discovery of related activity groups. The suggested solution interprets the challenge as a clustering problem. Utilizing a set of real world process models, we contrast activity clustering and existing structural model decomposition. The latter approach and its evaluation demonstrates the need for handling arbitrary activity grouping.

The remainder of the article is structured accordingly. Section 2 motivates the problem and introduces basic concepts. Section 3 presents the developed BPMA technique including the derivation of an abstract model behavioral profile and the model synthesis. Section 4 elaborates on the discovery of related activity groups. Section 5 provides empirical insights: we briefly describe a software application that implements the developed abstraction approach and present a case study. Section 6 discusses the related work, before Section 7 concludes and outlines the future work.

2. Background

This section discusses BPMA and explains the limitations of the existing approaches using an example. We also introduce the formalism used in the article.

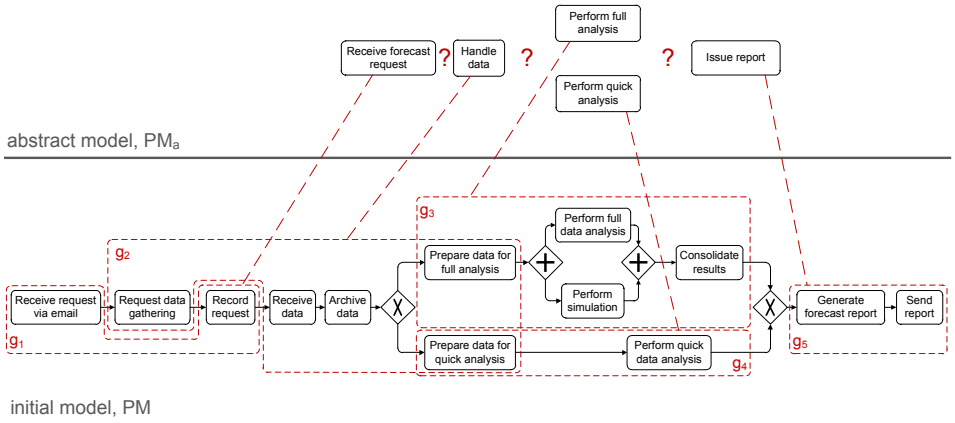


Fig. 1. Motivating example: initial model and activity grouping

2.1. *Business Process Model Abstraction*

Business process model abstraction is an operation on a model which preserves essential process properties by leaving out insignificant details in order to retain information relevant for a particular purpose. BPMA is realized by means of two basic abstraction operations: elimination and aggregation (respectively, inverse of the extension and refinement operations for behavior inheritance [38]). While elimination omits insignificant activities, aggregation groups several semantically related activities into one high-level activity.

Existing approaches restrict the choice of activities to be aggregated and derive the ordering relations between high-level activities analyzing the initial model control flow, cf. [8,23,34]. In these works each coarse-grained activity is mapped to a process model fragment. The fragments are either explicitly given by patterns or specified through properties. The latter enables aggregation of fragments with an arbitrary inner structure and rests upon model decomposition techniques [34]. However, the direct analysis of the control flow has limitations. In practice, semantically related activities can be allocated in the original model independently of the control flow structure, while one activity may belong to several semantically related activity sets [35,48]. Consider the example model PM in Fig. 1. The model describes a business process, where a forecast request is processed. Once a forecast request is received via email, a data gathering is requested and the request is recorded. Once the data is received, it is archived. Then, there are two options: either to perform a full data analysis, or its short version. The process concludes with a forecast report creation and dispatch. Model PM contains several semantically related activities that can be aggregated together into more coarse-grained ones. In model PM related activities are grouped by shapes with a dashed border, for instance, {*Prepare data for quick analysis, Perform quick data analysis*}. Each activity set corresponds to a respective high-level activity in the abstract model PM_a, e.g., *Perform quick analysis*. The abstraction techniques proposed in prior research permit the aggregation of

activities that belong, for instance, to groups g_3 and g_4. However, none of the existing approaches is capable of suggesting the ordering constraints between activities *Handle data*, *Perform full analysis*, and *Perform quick analysis* in the abstract model. In this article, we define a more flexible approach to determine the control flow structure. We utilize *behavioral profiles* [49] as the underlying formalism. Behavioral profiles capture behavioral characteristics of a process model in terms of strict order, exclusiveness, and interleaving order relations for each activity pair.

Further, addressing the user demand revealed in [40], our BPMA technique comprises a slider control that manages the ordering constraints loss. The slider allows the user to select an appropriate model from a spectrum of models: from the model with an arbitrary execution of high-level activities to the model where the ordering constraints of PM are best-effort preserved. Although emphasizing abstract model synthesis, we also exemplify how groups of semantically related activities can be identified. In this way, we extend this existing body of knowledge in this area, e.g., [14,39,41].

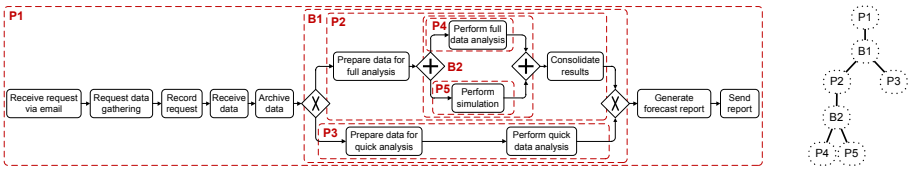
2.2. *Preliminaries*

For our discussion, we use a formal notion of a process model. It captures the commonalities of process modeling languages, such as BPMN or EPCs.

Definition 2.1. (Process Model)
A tuple $\mathsf{PM} = (A, G, F, t, s, e)$ is a *process model*, where:

- A is a finite nonempty set of activities;
- G is a finite set of gateways;
- $N = A \cup G$ is a finite set of nodes with $A \cap G = \emptyset$;
- $F \subseteq N \times N$ is the flow relation, such that (N, F) is a connected graph;
- $\bullet n = \{n' \in N | (n', n) \in F\}$ and $n\bullet = \{n' \in N | (n, n') \in F\}$ denote, respectively, the direct predecessors and successors of a node $n \in N$;
- $\forall\ a \in A : |\bullet a| \leq 1 \wedge |a \bullet| \leq 1$;
- $\forall\ g \in G : (|\bullet g| = 1 \wedge |g \bullet| > 1) \vee (|\bullet g| > 1 \wedge |g \bullet| = 1)$;
- $s \in A$ is the only start activity, such that $\bullet s = \emptyset$;
- $e \in A$ is the only end activity, such that $e\bullet = \emptyset$;
- $t : G \rightarrow \{AND, XOR\}$ is a mapping that associates each gateway with a type.

The execution semantics of a process model is given by a translation into a Petri net following on common formalizations [1,15]. As a process model has a dedicated start activity and a dedicated end activity, the resulting Petri net is a workflow net (WF-net) [1]. All gateways are of type AND or XOR, such that the WF-net is free-choice [1]. Then, the semantics of a process model follows from the Petri net formalization. We assume an interpretation of semantics in terms of a set of all complete traces (or execution sequences) from start s to end e. The set of *complete process traces* $\mathcal{T}_{\mathsf{PM}}$ for a process model PM contains (potentially infinitely many) lists of the form $s \cdot A^* \cdot e$ such that a list comprises the execution order of activities.



(a) (b)

Fig. 2. (a) The example process model PM with the fragments obtained by the RPST. (b) The RPST of PM.

We use $a \in \sigma$ with $\sigma \in \mathcal{T}_{\mathsf{PM}}$ to denote that an activity a is a part of a complete process trace.

Our approach also incorporates a structural decomposition of a process model, known as the Refined Process Structure Tree (RPST) [47]. The RPST parses a process model into a hierarchy of fragments, each having a single entry node and a single exit node. The containment hierarchy of these fragments forms a tree structure, the RPST. There are four different types of fragments. Single flows form trivial fragments, sequences of nodes (or fragments) of arbitrary length form polygon fragments, and a collection of fragments with shared boundary nodes forms a bond fragment. Any other structure is considered to be a rigid fragment. Note that the RPST is canonical and can be computed in linear time to the size of the process model.

We illustrate the RPST for the example model PM in Fig. 2. Fig. 2(a) highlights all fragments of the model. For instance, we observe a polygon fragment $P2$ that is defined by eight flows on the upper branch between the two XOR gateways. This fragment contains a bond fragment $B2$ that is defined by four flows between the AND gateways. Fig. 2(b) visualizes the containment hierarchy of the fragments, the RPST of the process model PM.

Our approach leverages the notion of a behavioral profile [49]. Such a profile provides an abstraction of trace semantics of a process model. It is based on *weak order* between activities. Two activities are in weak order, if there exists a complete trace in which one activity occurs after the other.

Definition 2.2. (Weak Order Relation)
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model, and $\mathcal{T}_{\mathsf{PM}}$ its set of traces. The *weak order relation* $\succ_{\mathsf{PM}} \subseteq (A \times A)$ contains all pairs (x, y), such that there is a trace $\sigma = n_1, \ldots, n_m$ in $\mathcal{T}_{\mathsf{PM}}$ with $j \in \{1, \ldots, m-1\}$ and $j < k \leq m$ for which holds $n_j = x$ and $n_k = y$.

Using weak order, we define three relations forming the behavioral profile.

Definition 2.3. (Behavioral Profile)
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model. A pair $(x, y) \in (A \times A)$ is in one of the following relations:

- strict order relation $\rightsquigarrow_{\mathsf{PM}}$, if $x \succ_{\mathsf{PM}} y$ and $y \not\succ_{\mathsf{PM}} x$;
- exclusiveness relation $+_{\mathsf{PM}}$, if $x \not\succ_{\mathsf{PM}} y$ and $y \not\succ_{\mathsf{PM}} x$;

- interleaving order relation $||_{\mathsf{PM}}$, if $x \succ_{\mathsf{PM}} y$ and $y \succ_{\mathsf{PM}} x$.

The set of all three relations is the *behavioral profile* of PM.

We illustrate behavioral profiles with the example model PM depicted in Fig. 1. Table 1 shows the relations for all pairs of activities. The relations of the behavioral profile, along with inverse strict order $\rightsquigarrow^{-1} = \{(x, y) \in (A \times A) \mid (y, x) \in \rightsquigarrow\}$, partition the Cartesian product of activities. Since exclusiveness and interleaving order are symmetric relations, whereas strict order is antisymmetric, Table 1 shows only half of the matrix. Further, we observe that an activity a is either exclusive to itself, i.e., $(a, a) \in +$, or in interleaving order with itself, i.e., $(a, a) \in ||$. The latter case is observed, once an activity may be observed more than once as part of a trace. This may be caused by a loop structure in the process model or a lack of synchronization.

Computation of the behavioral profile of a process model is done efficiently under the assumption of soundness. Soundness is a correctness criteria often used for process models that guarantees the absence of behavioral anomalies, such as deadlocks or livelocks [2]. It has been defined for WF-nets. Since we define semantics for process models by a translation into free-choice WF-nets, the soundness criterion can be directly applied to a process model. Further, we are able to reuse techniques for the computation of behavioral profiles introduced for sound free-choice WF-nets in [49]. Those allow for derivation of the behavioral profile in $O(n^3)$ time with n as the number of nodes of the WF-net. In principle, those techniques exploit the close relation between syntax and semantics of sound free-choice net systems. That is, interleaving order is traced back to concurrent enabling and control flow cycles,

	RE	*RDQ*	*RR*	*RD*	*AD*	*PDFA*	*PFDA*	*PS*	*CR*	*PDQA*	*PQDA*	*GFR*	*SR*
RE	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
RDG		$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
RR			$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
RD				$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
AD					$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
PDFA						$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
PFDA							$+_{\mathsf{PM}}$	$||_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
PS								$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
CR									$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
PDQA										$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
PQDA											$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
GFR												$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
SR													$+_{\mathsf{PM}}$

Table 1. The behavioral profile of model PM in Fig. 1

whereas strict order is deduced from the existence of a path between two activities that are not part of a common control flow cycle. Exclusiveness is deduced from the absence of a path between two activities in either direction.

Another approach for the computation of behavioral profiles exploits the RPST as discussed earlier, see [50]. It introduces the WF-tree for sound free-choice WF-nets, which is an annotated version of the RPST. The WF-tree assigns types to bond fragments (AND-type, acyclic XOR-type, loop). For bonds that are of type loop, it also captures the direction of the children fragments. The WF-tree allows for the computation of the behavioral profile for a pair of nodes based on their lowest common ancestor (LCA) fragment. For instance, if the LCA fragment is an acyclic bond fragment of type XOR (and the LCA fragment is not part of a bond fragment of type loop), we conclude that the nodes show exclusiveness according to the behavioral profile.

The behavioral profile relations capture different levels of execution freedom for activities. Interleaving order allows for the occurrence of activities in an arbitrary order, (inverse) strict order specifies a particular execution order, and exclusiveness prohibits occurrence of two activities in one trace. Accordingly, interleaving order can be seen as the absence of any order constraint. We organize the relations into a hierarchy presented in Fig. 3. At the top of the hierarchy the "strictest" relation appears, while at the bottom the least restrictive.

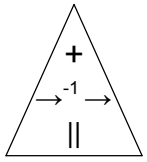


Fig. 3. Behavioral relation hierarchy

As mentioned earlier, the behavioral profile is an abstraction of trace semantics. Hence, equivalence of behavioral profiles does not imply trace equivalence of two process models. In particular, the profile captures the order constraints between *all* occurrences of two activities. Therefore, interleaving order may stem from two activities being part of a loop or from their concurrent enabling. In both cases, we do not observe a distinct order between all occurrences of two activities. As such, behavioral profiles are a coarse-grained abstraction that focus on the overall order of activities. Consequently, the detail of behavioral information considered by our abstraction approach depends on how precisely the profile approximates trace semantics of the process model. The larger the amount of interleaving order constraints, the less distinct order constraints are leveraged constructing the abstract model.

3. Abstract Model Synthesis

In this section, we describe the developed abstraction technique. We realize the technique in the following steps.

Step 1 derive the behavioral profile BP_{PM} for the initial model PM
Step 2 construct the behavioral profile BP_{PM_a} for the abstract model PM_a
Step 3 **if** a model consistent with profile BP_{PM_a} exists
Step 4 **then** create PM_a, **else** report an inconsistency.

In the previous section, we outlined how behavioral profiles are derived for a process model. The remainder of this section discusses steps 2 to 4 in detail.

3.1. *Abstract Model Behavioral Profile Construction*

We assume that each high-level activity in $\mathsf{PM}_a = (A_a, G_a, F_a, t_a, s_a, e_a)$ is the result of aggregation of several activities in $\mathsf{PM} = (A, G, F, t, s, e)$. Then, the construction of coarse-grained activities is formalized by the function *aggregate*:

Definition 3.1. (Function Aggregate)
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model and $\mathsf{PM}_a = (A_a, G_a, F_a, t_a, s_a, e_a)$ its abstract counterpart. Function $aggregate : \mathcal{A}_a \to (\mathcal{P}(A)\backslash\emptyset)$ specifies a correspondence between one activity in PM_a and the set of activities in PM.

Considering the example in Fig. 1, for instance, it holds *aggregate(Perform quick analysis)* = {*Prepare data for quick analysis, Perform quick data analysis*} and *aggregate(Handle data)*={*Collect data, Archive data, Prepare data for full analysis, Prepare data for quick analysis*}. The behavioral profile of model PM_a defines the relations between each pair of activities in PM_a. To discover the behavioral profile for PM_a we analyze the relations among activities in PM and consider the function *aggregate*. For each pair of coarse-grained activities x, y, where $x, y \in A_a$, we study the relations between a and b, where $(a, b) \in aggregate(x) \times aggregate(y)$. This study reveals a dominating behavioral relation between elements of $aggregate(x)$ and $aggregate(y)$. We assume that the behavioral relations between activity pairs of PM_a can be discovered independently from each other, i.e., the relation between x and y, where $x, y \in A_a$ depends on the relations between activities in $aggregate(x)$ and $aggregate(y)$, but does not depend on the relations between $aggregate(x)$ and $aggregate(z), \forall z \in A_a$, where $z \neq x$ and $z \neq y$.

Algorithm 3.1 formalizes the derivation of behavioral relations. The input of the algorithm is a pair of activities, x and y, and w_t—the user-specified threshold telling significant relation weights from the rest and, hence, managing the ordering constraints loss. The output of the algorithm is the behavioral profile relation between x and y. Algorithm 3.1 derives behavioral profile relations between x and y from the observable frequencies of relations between activities a and b, where $(a, b) \in aggregate(x) \times aggregate(y)$. According to Definition 2.3 each of the behavioral profile relations is specified by the corresponding weak order relations. Thereby, to conclude about the behavioral profile relation between x and y, we first evaluate the frequencies of weak order relations for x and y. The latter are found in the assumption that each weak order relation holding for $(a, b) \in aggregate(x) \times aggregate(y)$, contributes to the weak order relation between x and y. This rationale helps to find the weight for each weak order relation between x and y (lines 2–5). The overall number of relations is stored in variable w_{prod} (line 6). Algorithm 3.1 continues finding the relative weight for each behavioral profile relation (lines 7–10). The relative weights of behavioral relations together with the relation hierarchy are

Algorithm 3.1 Derivation of a behavioral relation for an activity pair

1: **deriveBehavioralRelation(Activity** x**, Activity** y**, Double** w_t**)**
2: $w(x \succ_{\mathsf{PM}_a} y) = |\{\forall (a,b) \in aggregate(x) \times aggregate(y) : a \rightsquigarrow_{\mathsf{PM}} b \vee a||_{\mathsf{PM}} b\}|$
3: $w(y \succ_{\mathsf{PM}_a} x) = |\{\forall (a,b) \in aggregate(x) \times aggregate(y) : a \rightsquigarrow^{-1}_{\mathsf{PM}} b \vee a||_{\mathsf{PM}} b\}|$
4: $w(x \not\succ_{\mathsf{PM}_a} y) = |\{\forall (a,b) \in aggregate(x) \times aggregate(y) : a \rightsquigarrow^{-1}_{\mathsf{PM}} b \vee a +_{\mathsf{PM}} b\}|$
5: $w(y \not\succ_{\mathsf{PM}_a} x) = |\{\forall (a,b) \in aggregate(x) \times aggregate(y) : a \rightsquigarrow_{\mathsf{PM}} b \vee a +_{\mathsf{PM}} b\}|$
6: $w_{prod} = |aggregate(x)| \cdot |aggregate(y)|$
7: $w(x +_{\mathsf{PM}_a} y) = \frac{min(w(x \not\succ_{\mathsf{PM}_a} y), w(y \not\succ_{\mathsf{PM}_a} x))}{w_{prod}}$
8: $w(x \rightsquigarrow_{\mathsf{PM}_a} y) = \frac{min(w(x \succ_{\mathsf{PM}_a} y), w(y \not\succ_{\mathsf{PM}_a} x))}{w_{prod}}$
9: $w(x \rightsquigarrow^{-1}_{\mathsf{PM}_a} y) = \frac{min(w(y \succ_{\mathsf{PM}_a} x), w(x \not\succ_{\mathsf{PM}_a} y))}{w_{prod}}$
10: $w(x||_{\mathsf{PM}_a} y) = \frac{min(w(x \succ_{\mathsf{PM}_a} y), w(y \succ_{\mathsf{PM}_a} x))}{w_{prod}}$
11: **if** $w(x +_{\mathsf{PM}_a} y) \geq w_t$ **then**
12: **return** $x +_{\mathsf{PM}_a} y$
13: **if** $w(x \rightsquigarrow_{\mathsf{PM}_a} y) \geq w_t$ **then**
14: **if** $w(x \rightsquigarrow^{-1}_{\mathsf{PM}_a} y) > w(x \rightsquigarrow_{\mathsf{PM}_a} y)$ **then**
15: **return** $x \rightsquigarrow^{-1}_{\mathsf{PM}_a} y$
16: **else**
17: **return** $x \rightsquigarrow_{\mathsf{PM}_a} y$
18: **if** $w(x \rightsquigarrow^{-1}_{\mathsf{PM}_a} y) \geq w_t$ **then**
19: **return** $x \rightsquigarrow^{-1}_{\mathsf{PM}_a} y$
20: **return** $x||_{\mathsf{PM}_a} y$

used to choose the dominating relation (lines 11–20). The behavioral relations are ranked according to their relative weights. Threshold w_t selects significant relations, omitting those for which the relative weights are less than w_t. Finally, the relation hierarchy allows us to choose the strictest relation among the significant ones. The input parameter w_t implements the slider concept: using w_t a user expresses the preferred ordering constraint loss level to obtain the respective behavioral relations for model PM_a.

To illustrate Algorithm 3.1 we refer to the example in Fig. 1 and derive the behavioral relations between activities of model PM_a. As before we acronym the names of activities. Assuming the threshold $w_t = 0.5$, abstraction results in an abstract model behavioral profile presented in Table 2. As relations of the behavioral profile are derived independently, we illustrate the construction of the behavioral profile in Table 2 looking at one activity pair. We elaborate on derivation of a behavioral relation for activities *Handle data* (HD) and *Perform quick analysis* (PQA). Following Algorithm 3.1, $w(HD \succ_{\mathsf{PM}_a} PQA) = 6$, $w(PQA \succ_{\mathsf{PM}_a} HD) = 0$, $w(HD \not\succ_{\mathsf{PM}_a} PQA) = 4$, $w(PQA \not\succ_{\mathsf{PM}_a} HD) = 10$, and $w_{prod} = 10$. Then, $w(HD +_{\mathsf{PM}_a} PQA) = 0.4$, $w(HD \rightsquigarrow_{\mathsf{PM}_a} PQA) = 0.6$, $w(HD \rightsquigarrow^{-1}_{\mathsf{PM}_a} PQA) = 0$, and $w(HD||_{\mathsf{PM}_a} PQA) = 0$. The constellation of behavioral relation weights is shown in Fig. 4. Each relation weight w_r defines a segment $[0, w_r]$, where the respective behavioral relation r is valid. If the

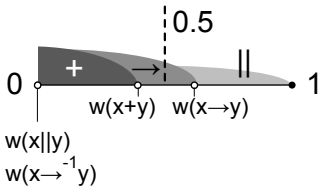


Fig. 4. Discovery of a behavioral relation for an activity pair *HD* and *PQA* of model PM_a Fig. 1. The weights of behabioral relations are evaluated according to the Algorithm 3.1, assuming $w_t = 0.5$.

maximum weight of the relations w_{max} is less than 1, we claim that the interleaving order relation is valid in segment $[w_{max}, 1]$ (it provides most freedom in execution of two activities). While the resulting segments overlap, the relation hierarchy defines the dominating relation in a particular point of $[0, 1]$. For $w(HD \rightsquigarrow_{\mathsf{PM}_a} PQA) \geq 0.5$ we state *Handle data* $\rightsquigarrow_{\mathsf{PM}_a}$ *Perform quick analysis* according to the behavioral relation hierarchy.

The Algorithm 3.1 terminates: it iterates over finite sets $aggregate(x)$ and $aggregate(y)$ and then compares the discovered relations. Given a pair of activities x and y in the abstract model PM_a, the time complexity of the Algorithm 3.1 is $O(k \cdot l)$, where $k = |aggregate(x)|$ and $l = |aggregate(y)|$. To construct the behavioral profile of model PM_a we need to derive the behavioral relations for each pair of activities in PM_a. Thereby, we need to investigate $\frac{|A_a|^2}{2}$ relations.

3.2. *Well-Structured Behavioral Profiles*

The creation of the behavioral profile as introduced above might yield a profile for which we cannot generate a process model. We use the notion of a *well-structured* behavioral profile to distinguish a class of behavioral profiles for which we can construct a process model. Whether a process model that satisfies the constraints of the behavioral profile exists depends on the applied notion of a process model and its structural and behavioral characteristics. For instance, the strict order relation may define a cyclic dependency between three activities a, b, and c: $a \rightsquigarrow b$, $b \rightsquigarrow c$, and $c \rightsquigarrow a$. The process model fragment in Fig. 5(a) satisfies these behavioral constraints at the expense of activity duplication. The result is clearly inappropriate against

	RFR	*HD*	*PFA*	*PQA*	*IR*
RFR	$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
HD		$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
PFA			$+_{\mathsf{PM}_a}$	$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
PQA				$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
IR					$+_{\mathsf{PM}_a}$

Table 2. The behavioral profile of PM_a constructed given model PM and function *aggregate* as informally defined in Fig. 1. The assumed weight threshold is $w_t = 0.5$.

(a) The fragment fulfills the constraints $a \rightsquigarrow b$, $b \rightsquigarrow c$, and $c \rightsquigarrow a$ at the expense of activity duplication.

(b) The fragment contains exactly one occurrence of a, b, and c. While the fragment complies with $a \rightsquigarrow b$ and $b \rightsquigarrow c$, it violates $c \rightsquigarrow a$.

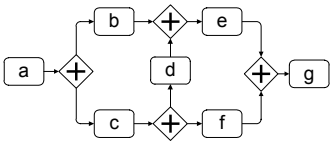
Fig. 5. Two process model fragments restricting the execution of a, b, and c

the background of our use case: an abstract model should provide a concise and compact view on the process. For our notion of a process model, the aforementioned behavioral constraints cannot be satisfied as exemplified by the model in Fig. 5(b), where $c \rightsquigarrow a$ is violated.

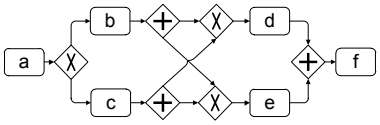
For the model synthesis, we focus on *well-structured* process models. Notice that our notion of a process model implies that models can be mapped to sound free-choice WF-nets. While soundness means the absence of behavioral anomalies, well-structuredeness refers to model topology. In a well-structured process model every split gateway has a corresponding join gateway, whereas both gateways bound a process model fragment with one entry node and one exit node [20]. The class of well-structured process models is of high practical importance. On the one hand, such models are easy to understand for humans [22]. On the other hand, well-structured process models can be efficiently handled by various analysis techniques, e.g., the computation of temporal constraints [10]. The class of well-structured process models is closely related to the RPST as discussed in Section 2.2.

Definition 3.2. (Well-Structured Process Model)
Let $\mathsf{PM} = (A, G, F, t, s, e)$ be a process model. The model PM is *well-structured*, iff the set of canonical components of the RPST of PM contains no rigid fragment.

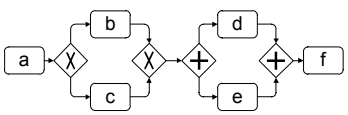
Fig. 6 exemplifies the notion of well-structured process models. Models PM_1 and PM_2 are not well-structured, as both contain rigids, while PM_3 is well-structured. Re-



(a) Model PM_1 is not well-structured



(b) Model PM_2 is not well-structured



(c) Model PM_3 is well-structured

Fig. 6. The process models PM_1 and PM_2 are not well-structured. While the process model PM_1 cannot be structured, PM_2 can be structured, resulting the behavior equivalent model PM_3.

(a) The order relations graph for process model PM_1

(b) The order relations graph for process models PM_2 and PM_3

Fig. 7. The order relations graphs of the models in Fig. 6

cently [31,32] developed an algorithm enabling process model structuring—construction of behaviorally equivalent well-structured process models for not well-structured process models. The behavioral equivalence is understood in terms of fully concurrent bisimulation [7]. However, not every process model can be structured. For instance, the algorithm of [31,32] delivers no well-structured process model that is behavior equivalent to PM_1. However, the algorithm structures model PM_2 delivering PM_3.

We design the synthesis of a well-structured process model following the structuring algorithm introduced by [31,32]. The structuring bases on the relations induced by a complete prefix unfolding and guarantees the preservation of a rather strong behavior equivalence. In the following, we show how the model synthesis defined for these relations is adapted to the behavioral profile relations.

To decide whether a well-structured process model can be constructed for a behavioral profile, we use the notion of an order relations graph. [31] introduced order relations graph capturing the order relations of a complete prefix unfolding. We construct an order relations graph for the behavioral profile relations. Doing so we reference the identity relation over activities in A as id_A.

Definition 3.3. (Order Relations Graph)
Let $\mathsf{BP} = \{\rightsquigarrow, +, ||\}$ be a behavioral profile over a finite set of activities A_{BP}. A tuple $g = (V, E)$ is an *order relations graph* of BP such that:

- $V = A_{\mathsf{BP}}$, i.e., the nodes are activities within A_{BP}
- $E = \rightsquigarrow \cup + \setminus id_{A_{\mathsf{BP}}}$, i.e., the edges correspond to the strict order relation and exclusiveness relation without self-relation of activities.

Edges in the order relations graph denote strict order and exclusiveness relations. We assume the strict order relation to be asymmetric and the exclusiveness relation to be symmetric. Thereafter, the strict order and exclusiveness relations are denoted in the graph as unidirectional or bidirectional edges, respectively. Fig. 7 shows the order relations graphs for the behavioral profiles of the models depicted in Fig. 6. As models PM_2 and PM_3 have equivalent behavior, they share one order relations graphs. That is due to the fact that the notion of equivalence assumed for structuring, fully concurrent bisimulation, is much stronger than behavioral profile equivalence, see [49].

The topology of a well-structured process model relates to the order relations graph structure. According to Definition 3.2 all the canonical components of the

RPST of a well-structured process model are of types trivial, polygon, or bond. Such components are represented in the order relations graph by node subsets that have uniform relations with all the remaining graph nodes. We refer to such node subsets as *modules*. Definition 3.4 formalizes the notion of a module and module types following [31].

Definition 3.4. (Module)
Let $g = (V, E)$ be an order relations graph.

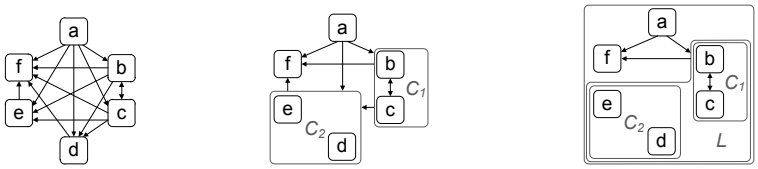
- A *module* $M \subseteq V$ is a non-empty set of nodes that have uniform relations with nodes in $V \setminus M$, i. e., $\forall\, x, y \in M, z \in (V \setminus M)$ it holds $(x, z) \in E \Leftrightarrow (y, z) \in E$ and $(z, x) \in E \Leftrightarrow (z, y) \in E$.
- Two modules $M, M' \subseteq V$ *overlap*, iff they intersect and neither is a subset of the other.
- A module $M \subseteq V$ is *strong*, iff there is no module $M' \subseteq V$, such that M and M' overlap.
- The empty set of nodes $\emptyset$, V, and the node sets of the from $\{v\}, \forall v \in V$ are *trivial* modules.
- A non-trivial module $M \subseteq V$ is *complete*, iff M induces the subgraph of g that is either complete or edgeless. If the subgraph is complete, we say that M is *XOR-complete*. If the subgraph is edgeless, we say that M is *AND-complete*.
- A non-trivial module $M \subseteq V$ is *linear*, iff there exists a linear order $(v_1, \ldots, v_{|M|})$ of elements of M, such that $(v_i, v_j) \in E$ and $(v_j, v_i) \notin E$ for $i, j \in \mathbb{N}$, $1 \leq i, j \leq |M|$ and $i < j$.
- A non-trivial module $M \subseteq V$ is *primitive*, iff it is neither complete nor linear.

To discover modules we leverage the modular decomposition [25]. Modular decomposition of a graph results in a unique arborescence of maximal non-overlapping modules.

Definition 3.5. (Modular Decomposition)
Let $g = (V, E)$ be an order relations graph. The *modular decomposition tree* is a tuple $\mathcal{MDT}_g = (\Omega, \chi)$, such that Ω is a set of all strong modules and $\chi : \Omega \to \mathcal{P}(\Omega)$ is a function that assigns child modules to modules with $\forall\, \omega, \gamma \in \Omega\ [\ (\chi(\omega) \cap \chi(\gamma) \neq \emptyset) \Rightarrow \omega = \gamma\ ]$.

Fig. 8 exemplifies the modular decomposition for the order relations graph in Fig. 7. The order relations graph is stepwise decomposed into a hierarchy of strong modules. Two sets of nodes $\{b, c\}$ and $\{d, e\}$ are identified as strong modules that have equal relations to all other nodes in the graph. Both modules together constitute another module, as the former modules are of equal relations to the nodes a and f.

The modular decomposition of an order relations graph characterizes behavioral profiles for which we construct an according well-structured process model. That is, we check for the absence of a primitive module in the modular decomposition. Note that we implicitly assume that the relational properties of a behavioral profile are satisfied.



(a) Order relations graph

(b) Complete modules C_1 and C_2 are discovered

(c) Nodes a and f with modules C_1 and C_2 constitute a linear module L

Fig. 8. The step-wise modular decomposition of an order relations graph. In the initial order relations graph node sets {b, c} and {d, e} are discovered as strong modules. Module C_1 is XOR-complete, while C_2 is AND-complete. Nodes a and f with modules C_1 and C_2 constitute linear module L.

Finally, we return to the illustrative example. Fig. 9 shows the order relations graph and its decomposition for the behavioral profile in Table 2.

Definition 3.6. (Well-Structured Behavioral Profile)
Let $\mathsf{BP} = \{\rightsquigarrow, +, ||\}$ be a behavioral profile over a finite set of activities A_{BP} and g—the order relations graph of BP. The behavioral profile BP is *well-structured*, iff the modular decomposition tree of g, $\mathcal{MDT}_g$, contains no primitive module.

In the example with the three activities a, b, and c, where $a \rightsquigarrow b$, $b \rightsquigarrow c$, and $c \rightsquigarrow a$ the profile is not well-structured. The modular decomposition of the respective order relations graph comprises a primitive module covering the three activities. Fig. 7 visualizes the order relations graphs for the process models in Fig. 6. The graph in 7(a) does not represent a well-structured behavioral profile since the modular decomposition tree contains a primitive module. The modular decomposition of the graph in Fig. 7(b) is shown in Fig. 8. As the modular decomposition contains no primitive module, graph in Fig. 7(b) represents a well-structured behavioral profile. We conclude that *1)* model PM_1 in Fig. 6 has a non-well-structured behavioral profile, and *2)* the behavioral profile of models PM_2 and PM_3 is well-structured. Returning to the process used as the illustrative example, consider Fig. 9. Fig. 9(a) shows the relations graph corresponding to the behavioral profile of the abstract process model. The modular decomposition of this order relations graph is presented in Fig. 9(b). The decomposition has two modules: the complete module C and the linear module L.

(a) The order relations graph for the behavioral profile in Table 2

(b) Decomposition of the order relations graph

Fig. 9. The order relations graphs for the behavioral profile in Table 2 and its modular decomposition

We use the existing methods for graph modular decomposition to decide if a behavioral profile is well-structured. Verification of a behavioral profile well-structuredeness can be done in linear time. According to Definition 3.6, we create the modular decomposition tree of the order relations graph of the validated behavioral profile. The modular decomposition tree can be constructed in linear time [25]. The number of strong modules in the modular decomposition tree is linear to the size of the graph [25].

Finally, we show that well-structuredness of a behavioral profile is a necessary condition for the existence of a well-structured process model exhibiting this profile.

Lemma 3.1. (Lemma 1) The behavioral profile of a sound well-structured process model is well-structured.

Proof. As we consider process models that can be mapped to sound free-choice WF-nets, we can rely on the WF-tree as discussed in Section 2.2. First, we construct the RPST and annotate it to get the WF-tree of the sound process model $\mathsf{PM} = (A, G, F, t, s, e)$. According to our notion of a process model, each activity is a boundary node of at most two trivial fragments of these trees. Let α and β be two trivial fragments for which the fragment entries are two distinct activities $a, b \in A$, $a \neq b$. Since PM is well-structured, the RPST and, therefore, also the WF-tree, does not contain any rigid fragment. Let γ be the lowest common ancestor (LCA) fragment of α and β in the WF-tree. According to the Proposition 4.1 in [50], the profile relation for activities a and b (in the absence of rigid fragments) can be deduced from *1)* the type of γ, and *2)* the existence of a loop fragment on the path from the root of the tree to γ. If the fragments α and β are part of a loop fragment, the corresponding module is *and*-complete. If they are not a part of the loop fragment, the type of γ determines the type of the module.

For any fragment of the WF-tree, there is a module in the respective modular decomposition tree $\mathcal{MDT}_g$ of the order relations graph g of the behavioral profile. A polygon yields a linear module, a bond fragment of type AND—an *and*-complete module, an acyclic bond fragment of type XOR—a *xor*-complete module. Hence, $\mathcal{MDT}_g$ does not contain any primitive module and the behavioral profile is well-structured. □

3.3. *Synthesis of a Process Model from a Well-Structured Behavioral Profile*

Once well-structuredness of a behavioral profile is verified, we proceed with the model synthesis. The synthesis algorithm iteratively constructs a model from the modules identified by the modular decomposition. We largely rely on the synthesis algorithm presented in [31,32].

Algorithm 3.2 outlines the steps of the model synthesis. First, we construct the order relations graph of the behavioral profile (line 1). Modular decomposition discovers modules in the order relations graph (line 2). The algorithm iterates

Algorithm 3.2 Synthesis of a sound well-structured process model from a well-structured behavioral profile

```
1: synthesizeModel(BP = {⇝, +, ||})
2: g = constructOrderRelationsGraph(b)
3: (Ω, χ) = modularDecomposition(g)
4: for all ω ∈ Ω following on a postorder traversal using χ do
5:    if ω is trivial then
6:       add activity to PM
7:    if ω is AND-complete then
8:       construct bond fragment of type AND in PM
9:    if ω is XOR-complete then
10:      construct acyclic bond fragment of type XOR in PM
11:   if ω is linear then
12:      construct trivial or polygon in PM
13: if PM misses start or end activity then
14:   add start and/or end activity to PM
15: for all a ∈ A_PM such that a||a do
16:   insert control flow cycle around a in PM
17: return PM
```

over all the identified modules to construct the model skeleton (lines 2–14). Trivial modules contribute only single activities to the model. A complete module leads to the creation of a bond fragment of type AND or an acyclic bond fragment of type XOR. Such a bond comprises all activities or model fragments encapsulated by this complete module. A linear module leads to the creation of a polygon connecting the respective activities or model fragments. If the resulting model structure is gateway-bordered, it is normalized to satisfy the structural requirements of the process model (lines 13–14). For all activities that have interleaving order as their self-relation according to the behavioral profile, we insert circuits into the created process model structure (lines 15–26). Those comprise a bond fragment of type loop and polygons. This transformation step is illustrated in Fig. 10. As the final step, the algorithm returns the process model.

We prove the correctness of the Algorithm 3.2 as follows.

Proposition 3.1. (Proposition) Algorithm 3.2 terminates and after termination the sound well-structured process model $\mathsf{PM} = (A, G, F, t, s, e)$ shows the behavioral profile used as the algorithm's input.

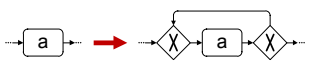


Fig. 10. Insertion of a bond fragment of type loop for an activity with interleaving order as a self-relation

Proof. First, we show that the resulting model is indeed a sound well-structured model. Second, we prove that the behavioral profile used as the input coincides with the behavioral profile of the created model for the respective activities.

Termination The set of activities of the behavioral profile is finite. Thus, the order relations graph and the number of modules identified in the decomposition are finite. Once we iterate over all activities and modules, the algorithm terminates.

Result We first prove the correctness of syntax, then of semantics. Finally, we consider the behavioral profile correctness.

Syntax The algorithm creates a process model $\mathsf{PM} = (A, G, F, t, s, e)$ by a postorder traversal of the modular decomposition tree. Hence, for all nodes there is a path from (to) the node that represents the entry (exit) of the component created for the root module. If those nodes are gateways, the algorithm adds a start and end activities. Hence, PM aligns with the process model notion described in Definition 2.1. As the algorithm constructs only trivial, polygon, and bond components (the trivial circuit is a bond fragment as well), model PM can be mapped to a free-choice WF-net.

Semantics As it follows from step Syntax the Algorithm 3.2 delivers a process model that can be mapped to a WF-net. The model is constructed by nesting trivial, polygon, and bond fragments. The constructed bond fragments are acyclic, either of type AND or XOR. The construction of a trivial circuit inserts polygons and bonds of type loop. Trivial and polygon fragments do not cause unsoundness. Further, Lemma 1 and Lemma 2 in [50] argue that place- and transition-bordered bonds do not cause unsoundness. As bonds of type XOR and AND are mapped, respectively, to place- and transition-bordered bonds the created model satisfies the soundness requirements. Thereafter, the produced process model is sound.

Behavioral Profile The constructed process model $\mathsf{PM} = (A, G, F, t, s, e)$ is sound, well-structured, and mappable to a WF-net. According to Lemma 3.1, the behavioral profile of PM is well-structured. This behavioral profile coincides with the behavioral profile used as the algorithm's input. The latter follows from the types of the constructed fragments. Neglecting the trivial circuits inserted at the end, the algorithm creates a trivial, polygon, or bond fragment depending on the type of the module, i.e., depending on the relations observed between the activities of the module in the order relations graph. For two distinct activities $a, b \in A$, $a \neq b$, this fragment is the LCA of the trivial fragments α, β for which the fragment entries are a and b in the WF-tree of the model, respectively. Neglecting the trivial circuits, the type of the LCA fragment determines the profile relation, see Proposition 1 in [50]. A trivial circuit includes only one activity causing the interleaving order as the self-relation for this activity. Therefore, an insertion of trivial circuits has no impact on the relation between two distinct activities. Trivial circuits are introduced only for activities with interleaving order as the self-relation. Thus, the behavioral profile of the constructed process model coincides with the behavioral profile used as the algorithm input. □

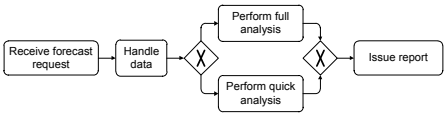


Fig. 11. Abstract representation of the process "Forecast request handling". This model is obtained from the model PM in Fig. 1 using the abstraction algorithm based on behavioral profiles with the threshold of 0.5 and the activity groups as defined in Fig. 1.

Corollary 3.1. Given a well-structured behavioral profile, the construction of a process model exhibiting this behavioral profile can be solved in linear time.

Proof. We represent the relations used in the process model synthesis as bi-dimensional arrays that map to zero or one. Against this background, adding an entry to a relation and checking a tuple membership is done in constant time. The construction of order relations graph takes linear time to the size of the behavioral profile. Further, the modular decomposition tree for order relations graph is realized in linear time [25]. We proceed iterating the strong modules in the modular decomposition tree. The number of strong modules is linear to the graph size [25]. The construction of a respective model fragment takes linear time to the behavioral profile size. Process model normalization implies the check for the start and end activities. This operation also takes linear time. Finally, insertion of trivial circuits takes linear time to the size of the behavioral profile. □

Now we can make the following statement.

Theorem 3.1. *There exists a sound well-structured free-choice WF-system, if and only if the behavioral profile is well-structured.*

Proof. $\Rightarrow$ follows from Lemma 3.1, $\Leftarrow$ from Proposition 3.1. □

We conclude this section returning to the motivating example presented in Fig. 1. Fig. 11 illustrates the complete abstract model derived from the initial model according to the developed abstraction technique. Following Algorithm 3.2 the model in Fig. 11 is obtained from the modular decomposition presented in Fig. 9.

4. Discovery of Semantically Related Activity Groups

This section elaborates on one approach for discovery of semantically related activity groups—activity clustering according to the process model data flow. We argue that activity group discovery can be interpreted as a cluster analysis problem. Further, among the existing clustering algorithms we select those that fit the abstraction use case best.

Modularity is one of the core system design principles [21,30]. It reflects the extent to which system components are independent of each other and can be recomposed. *Cohesion* and *coupling* are key metrics to assess the quality of modularization in system design [43]. While cohesion shows how strongly the functionality of one

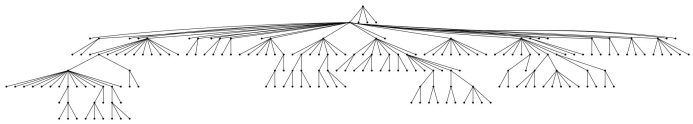
module is related, coupling indicates the number of intermodule dependencies. A good system design implies high cohesion of functionality inside a module and low coupling between modules. Coupling and cohesion are also used in business process modeling. The key idea behind this adoption is based on two observations with regard to business process activity definition [36]. First, elementary operations with intensive data flow between them (high cohesion) are good candidates for being aggregated. Second, the grouping of elementary operations into business activities must result in low intergroup data flow, i.e., low activity coupling.

Building upon this idea, we discover activity groups (to be aggregated into coarse-grained activities) analyzing process model data flow. We assume that groups of related activities operate on the same data objects. Hence, given the initial process model enriched with data flow, we seek for activity groups that fulfill two requirements. Information flow inside groups is dense showing high cohesion. Intergroup information flow is sparse, i.e., groups are loosely coupled.

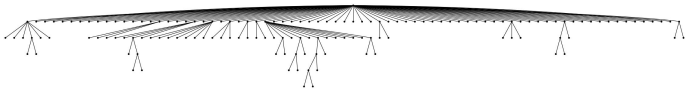
To arrive at such groups we use cluster analysis. The employed activity clustering algorithm proceeds as follows. First, we construct a data flow graph. The vertices of the data flow graph are process model activities. An edge connects two activities, once one activity produces a data object, while the other consumes this data object. If two activities operate on more than one data object, they are connected by multiple edges, one edge per data object. The data flow graph is partitioned using min-cut algorithm proposed by Stoer and Wagner in [44]. This algorithm takes as the input the graph and partitions it into two subgraphs, so that the number of edges connecting the vertices of two subgraphs is minimal. The min-cut algorithm is applied iteratively. In the first step the partitioning input is the data flow graph. The next iteration partitions two subgraphs of the initial data flow graph. The partitioning process continues until subgraphs with one vertex are obtained.

The result of iterative graph partitioning is captured as a *dendrogram*—a tree capturing the hierarchy of activity clusters. A node of the dendrogram corresponds to an activity cluster, i.e., the set of vertices in a graph. Two dendrogram nodes n_1 and n_2 are connected to node n, if partitioning of the activity cluster corresponding to node n produces two clusters corresponding to nodes n_1 and n_2. Whereas the root of the dendrogram corresponds to the set of process model activities, its leaves correspond to activities. The deeper in the dendrogram the node is, the higher is the cohesion between the activities of the corresponding activity cluster.

Within partitioning a graph may have several minimal cuts of the same weight. Thus, such graphs can be partitioned in various alternative ways. In this case the partitioning algorithm allocates activity clusters with the same coupling values at various depth in the dendrogram. However, we are interested not in the actual graph partitioning order, but in the values of activity group coupling, i.e., minimal cut values. Therefore, we postprocess the dendrogram so that the minimal cut value always increases with the increase of the graph depth. Fig. 12(b) presents an example of such a dendrogram. The dendrogram helps to select the desired activity granularity. Given a dendrogram, the user specifies a horizontal cut in it. The cut results in a



(a) RPST



(b) Dendrogram

Fig. 12. Comparison of the RPST and dendrogram of the process model PM_1

new tree. The leaves of this tree become activities of the abstract process model. As each node corresponds to an activity cluster, the new tree provides information about the resulting activity clustering.

5. Evaluation

In this section, we focus on the applicability of the presented abstraction technique. First, we shortly present the tool Flexab, which provides an implementation of flexible model abstraction. Then, we report on the findings of applying the abstraction technique in a case study.

5.1. *Implementation*

Flexab realizes flexible process model abstraction for process models defined as Petri nets. The tool has been discussed in detail in [51], so that we restrict the discussion to its core functionality at this stage. Flexab builds upon the Oryx framework [12].

Oryx is an extensible modeling framework that is completely web-based. The Oryx framework comprises an editor to create process models in various notations, a repository to manage process model collections, and a mashup framework to realize additional functionality concerning multiple process models. Our approach to process model abstraction has been realized using the Oryx mashup framework. On the client side, the Oryx mashup framework already consists of gadgets that allow to view a process model and to select elements of a process model. To realize the abstraction approach, a Flexab gadget has been implemented. It allows to define multiple of groups of process model elements that shall be aggregated in the course of abstraction. Once all activity groups have been defined and associated with a label for the derived aggregated activity, the actual abstraction is triggered. The abstraction approach has been realized as a server side component in the backend of the Oryx mashup framework. Thus, the Flexab gadget calls a servlet that conducts the abstraction according to the defined activity groups. Then, the servlet relies on further Oryx components to store the abstracted model in the Oryx repository and to render a graphical representation. Once this representation is available, another gadget is opened in the Oryx mashup framework to display the abstraction result.

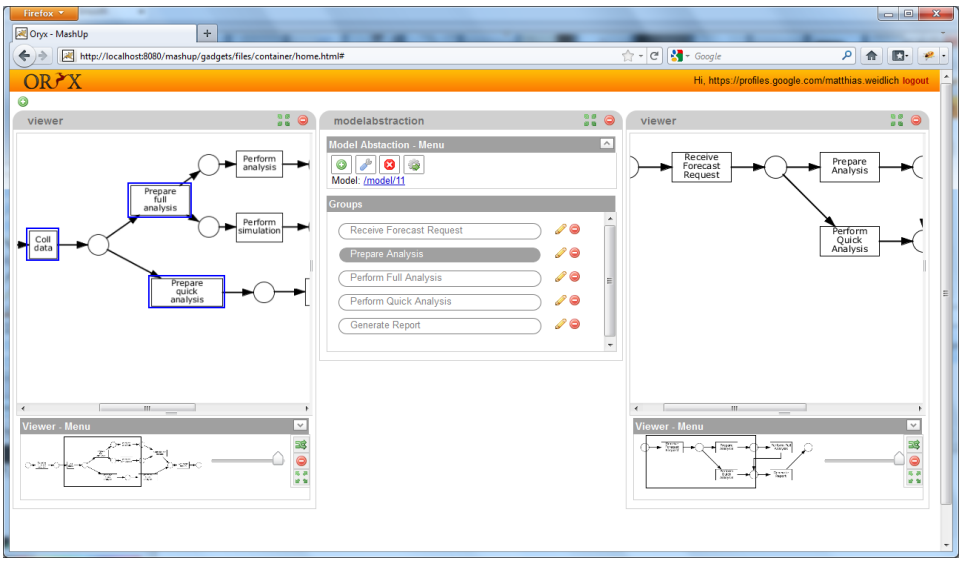


Fig. 13. Flexab, an implementation of flexible process model abstraction based on the Oryx framework

We illustrate the functionality of Flexab using the screenshot depicted in Fig. 13. It shows the Oryx mashup framework with three gadgets. The viewer gadget on the left hand side depicts an example process model defined as a Petri net. It has been created using the Oryx editor and is stored in the Oryx repository. The Flexab gadget is shown in the middle. It allows for grouping activities that shall be abstracted. In the screenshot, one group is selected. All activities that are contained in this group are highlighted in the viewer gadget on the left hand side. The Flexab gadget provides a button to trigger abstraction. The result is presented in another viewer gadget. In the screenshot, this gadget is located on the right hand side.

5.2. *Case Study*

This section presents an empirical study that emphasizes the added value of the abstraction advocated in this article. To support our argument we use a collection of real world process models. First, we leverage this collection to witness the limitations of structural abstraction methods and motivate the demand for the introduced abstraction. Afterwards, we use one model from this set to illustrate the application of the novel abstraction method in an industrial setting.

As mentioned before, numerous BPMA approaches rely solely on the process model structure [8,23,33,34]. For instance, the decomposition of a process model into single entry and single exit (SESE) fragments is hierarchical and can be represented as a *process structure tree* or PST. However, the abstraction based on behavioral profiles provides more reach capabilities. To motivate the need for these capabilities we begin this section illustrating the limitations of structural methods. This section uses an empirical argument and studies the set of industrial process models. We

consider the subset of 36 largest process models of the SAP Reference Model [19]—the collection that has been used in several works on process model analysis. The SAP Reference Model captures business processes that are supported by the SAP R/3 software in its version from the year 2000. It is organized in 29 functional branches of an enterprise, like sales or accounting, covered by the SAP software. We select the largest process models in terms of size: the inspected models contain from 21 to 130 nodes with a number of activities varying between 9 and 43.

We inspect the process models and identify activity groups according to the model data flow, see Section 4. Once the groups are obtained, we compare them with the groups delivered by the process model decomposition. For each process model we observe two parameters: the total number of groups and the groups that do not constitute a SESE fragment. The outcome of this study shows that 19.5% of the discovered activity groups do not form SESE fragments. This observation is in line with the outcomes of earlier studies, see, for instance, [35]. Altogether, this finding emphasizes the need for techniques that are able to handle arbitrary groupings of activities in an abstraction scenario.

The remainder of this section illustrates the application of the developed approach by the example of one industrial process model. For this purpose we select one model of the 36 models studied. The model in question captures the business process *Procurement of Materials and External Services*. The model has reasonable size: it contains 23 nodes with 9 activities among them. This means that it aligns well with the existing modeling guidelines [6,27] and can be easily understood by a human reader. Yet, we select exactly this model for the case study, as it vividly illustrates the capabilities of our approach. The choice of a more complex model would impede the illustration. Fig. 14 shows the original process model as an EPC. This model can be transformed into a process model according to Definition 2.1, yielding the model PM in Fig. 15 with the same trace semantics.

The study of model PM shows that several activities in the model make use of the same data objects. For instance, activities *Processing of shipping notification* and *Transmission of shipping notification* operate with one object *Shipping notification*. Accordingly, we group those activities that operate on the same data objects following the mechanism introduced in Section 4. This gives us three activity groups, g_1, g_2, and g_3, marked in the figure by the shapes with dashed borders. Three activities are not grouped and remain as is: each of them operates on a separate data object. Notice that activities of the group g_2 do not constitute a SESE fragment. Thereafter, the input for abstraction algorithm is the initial process model enriched with information about activity grouping. The case study illustrates the capabilities of our abstraction approach by two abstractions that differ in the threshold values. One scenario uses the threshold value of 0.2, while the other scenario uses 0.5 value.

The first step of the abstraction algorithm is the synthesis of the behavioral profile for model PM. Table 3 presents the corresponding profile BP. Once we have obtained the profile BP and the information about activity grouping, we derive the

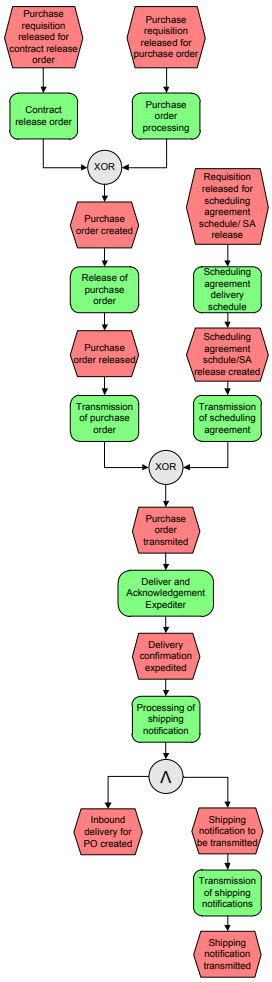


Fig. 14. Motivating example: the initial model of business process *Procurement of Materials and External Services*

behavioral profile of the abstract model according to Algorithm 3.1. This algorithm is also parametrized by the user-specified threshold value. Since the case study considers two threshold values, we arrive at two behavioral profiles. In the case of 0.2 threshold value we obtain the behavioral profile presented in Table 4(a), while in the case of 0.5 threshold value, the behavioral profile presented in Table 4(b). The profiles have one difference: the relation between activities *Contract release order* and *Creation and delivery of purchase order*. According to the Algorithm 3.1 the threshold value of 0.2 results in the exclusiveness relation, while the 0.5 value causes the strict order relation.

Subsequently, the model synthesis brings us to the two process models: one for each behavioral profile. The model PM_a in Fig. 15 is synthesized from the behavioral profile in Table 4(a), while the model PM'_a corresponds to the behavioral profile in

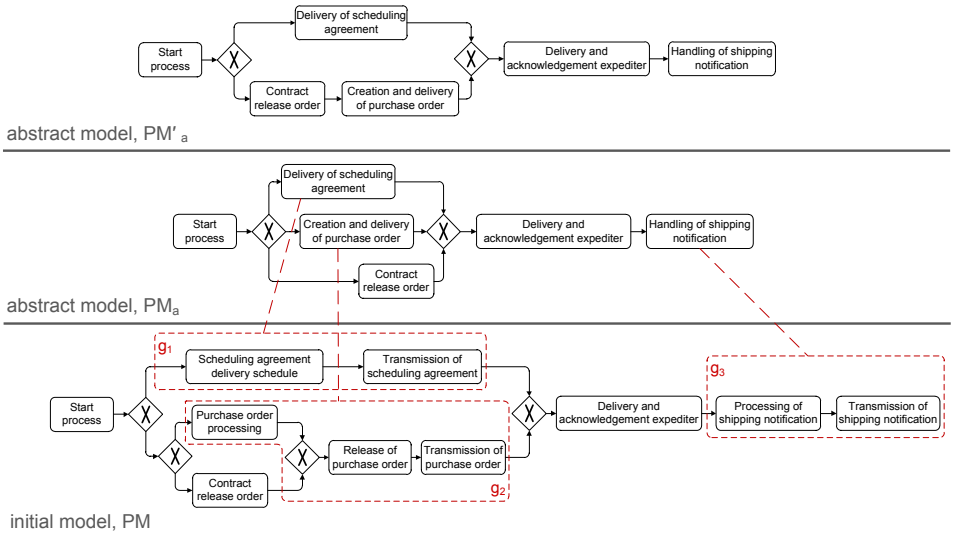


Fig. 15. The initial process model PM and two models, PM_a and PM'_a, abstracting it. The model PM is enriched with activity grouping information. The model PM_a is obtained from PM given the abstraction threshold value of 0.2, while PM'_a is obtained with the threshold value of 0.5.

Table 4(b). As the two profiles differ in one behavioral relation, the models vary accordingly. Once we have the abstract process models available, we compare them with the initial model PM. In particular, we are interested in the ordering constraint between the activities *Contract release order* and *Creation and delivery of purchase order*, strict order or exclusiveness. These two alternative relations stem from the relations between activity *Contract release order* and activities of the group g_2 in PM, where both the strict order and exclusiveness take place. In this way the newly introduced abstraction approach summarizes the ordering constraints of the initial process model even in the case of non-hierarchical abstraction, but at the price of high ordering constraints loss.

6. Related Work

The work presented in this article complements two research areas: business process model abstraction and process model synthesis. The former studies methods of process model transformation and criteria of model element abstraction.

The related process model transformation techniques constitute two groups. The first group builds on an explicit definition of a fragment to be transformed. Here, Petri net reduction rules preserving certain behavioral properties play an important role [28]. Such rules have also been defined for workflow graphs [37], EPCs [16,26], and YAWL [52]. The second group of transformation techniques hierarchically decomposes a model into fragments, e.g., cf. [47]. The reduced process model can be regarded as a view in terms of [29] and typically preserves properties of behavior inheritance [3]. Unfortunately, such hierarchical decomposition is not sufficient in many scenarios, cf. [48]. The

	SP	*SADS*	*TOSE*	*POP*	*ROPO*	*TOPO*	*CRO*	*DAAE*	*POSN*	*TOSN*
SP	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
SADS		$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
TOSE			$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
POP				$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
ROPO					$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
TOPO						$+_{\mathsf{PM}}$	$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
CRO							$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
DAAE								$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
POSN									$+_{\mathsf{PM}}$	$\rightsquigarrow_{\mathsf{PM}}$
TOSN										$+_{\mathsf{PM}}$

Table 3. The behavioral profile of the process model PM

technique developed in this article shows how the abstract model control flow can be discovered even for non-hierarchical abstractions. Model element abstraction criteria, for instance, execution cost, duration, and path frequency, have been studied in a number of works [17,18,40]. These works have in common that their major focus is on identifying abstraction candidates. The current article complements this stream of research demonstrating how abstracted process models can be constructed even if aggregated activities are not structurally close to each other. There is a series of works that address the requirements of business process model abstraction. The approaches of [8,9,23] build on an explicit definition of a fragment that can be abstracted to provide a process overview. In [5,34,45] such fragments are discovered without user specification according to the model structure.

The developed method for the construction of an abstract process model from the behavioral profile extends the family of process model synthesis techniques. In process mining the alpha algorithm is used for the construction of a process model from event logs [4]. The mining relations used by the alpha algorithm differ to ours

(a) Behavioral profile with the threshold 0.2

	SP	*DOSA*	*CADOPO*	*CRO*	*DAAE*	*HOSN*
SP	$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
DOSA		$+_{\mathsf{PM}_a}$	$+_{\mathsf{PM}_a}$	$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
CADOPO			$+_{\mathsf{PM}_a}$	$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
CRO				$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
DAAE					$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
HOSN						$+_{\mathsf{PM}_a}$

(b) Behavioral profile with the threshold 0.5

	SP	*DOSA*	*CADOPO*	*CRO*	*DAAE*	*HOSN*
SP	$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
DOSA		$+_{\mathsf{PM}_a}$	$+_{\mathsf{PM}_a}$	$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
CADOPO			$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
CRO				$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
DAAE					$+_{\mathsf{PM}_a}$	$\rightsquigarrow_{\mathsf{PM}_a}$
HOSN						$+_{\mathsf{PM}_a}$

Table 4. The behavioral profiles of abstract process models

as they are only partially transitive. In this article, we use the behavioral profile relations, which permit the reconstruction of the process model if the profile is consistent. Several extensions of the alpha algorithm have been proposed to deal with loops, among others using relations that capture transitive relations similar to the weak order of behavioral profiles [46]. The idea of considering probabilities of uncertain relations in our approach is also partially inspired by process mining [18]. Finally, there is further work on synthesis that take the state space as an input to generate Petri net process models including [11,13,24].

7. Conclusion and Future Work

This article has presented a novel approach to process model abstraction for the case in which activities can be arbitrarily grouped. To this end we use the behavioral profiles and a notion of well-structuredness. First, we derive the behavioral profile of the initial model. Then, this profile is abstracted using a derivation algorithm. If the abstract profile is well-structured, we can guarantee that the process model to be generated from it is sound. Beyond that, we analyze different options of aggregating activities. We define a clustering approach that works on the data flow of a process model. The evaluation of the clustering results and its comparison to structural aggregation emphasizes the need to provide abstraction techniques that are able to handle arbitrary groupings of activities.

The reported research motivates several directions of the future work. While we suggest one approach to activity group discovery, criteria and methods enabling activity grouping call for deeper investigation. We aim to compare our with further modularization techniques as proposed in [35]. Another direction of future work relates to model synthesis out of behavioral profiles. In particular, we want to use a negative well-structuredness analysis for identifying its closest well-structured profile.

References

1. W. M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.
2. W. M. P. van der Aalst. Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques. In *BPM 2000*, volume 1806 of *LNCS*, pages 161–183, 2000.
3. W. M. P. van der Aalst and T. Basten. Life-Cycle Inheritance: A Petri-Net-Based Approach. In *ICATPN 1997*, pages 62–81, London, UK, 1997. Springer.
4. W. M. P. van der Aalst, A. J. M. M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
5. A. Basu and R.W. Blanning. Synthesis and Decomposition of Processes in Organizations. *Information Systems Research*, 14(4):337–355, 2003.
6. J. Becker, M. Rosemann, and Ch. von Uthmann. Guidelines of Business Process Modeling. In *BPM 2000*, volume 1806 of *LNCS*, pages 30–49. Springer, 2000.
7. E. Best, R. R. Devillers, A. Kiehn, and L. Pomello. Concurrent Bisimulations in Petri Nets. *Acta Informatica*, 28(3):231–264, 1991.

8. R. Bobrik, M. Reichert, and T. Bauer. View-Based Process Visualization. In *BPM 2007*, volume 4714 of *LNCS*, pages 88–95, Berlin, 2007. Springer.
9. J. Cardoso, J. Miller, A. Sheth, and J. Arnold. Modeling Quality of Service for Workflows and Web Service Processes. Technical report, University of Georgia, 2002. Web Services.
10. C. Combi and R. Posenato. Controllability in Temporal Conceptual Workflow Schemata. In *BPM 2009*, volume 5701 of *LNCS*, pages 64–79. Springer, 2009.
11. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 47(8):859–882, August 1998.
12. G. Decker, H. Overdick, and M. Weske. Oryx—An Open Modeling Platform for the BPM Community. In *BPM 2008*, pages 382–385, 2008.
13. J. Dehnert and W. M. P. van der Aalst. Bridging The Gap Between Business Models And Workflow Specifications. *International Journal of Cooperative Information Systems*, 13(3):289–332, 2004.
14. C. Di Francescomarino, A. Marchetto, and P. Tonella. Cluster-based Modularization of Processes Recovered from Web Applications. *Journal of Software Maintenance and Evolution: Research and Practice*, 2010.
15. R. M. Dijkman, M. Dumas, and Ch. Ouyang. Semantics and Analysis of Business Process Models in BPMN. *Information & Software Technology*, 50(12):1281–1294, 2008.
16. B. F. van Dongen, M. Jansen-Vullers, H. Verbeek, and W. M. P. van der Aalst. Verification of the SAP Reference Models Using EPC Reduction, State-space Analysis, and Invariants. *Computers in Industry*, 58(6):578–601, 2007.
17. R. Eshuis and P. Grefen. Constructing Customized Process Views. *Data & Knowledge Engineering*, 64(2):419–438, 2008.
18. C. W. Günther and W. M. P. van der Aalst. Fuzzy Mining-Adaptive Process Simplification Based on Multi-perspective Metrics. In *BPM 2007*, volume 4714 of *LNCS*, pages 328–343, Berlin, 2007. Springer.
19. G. Keller and T. Teufel. *SAP(R) R/3 Process Oriented Implementation: Iterative Process Prototyping*. Addison-Wesley, 1998.
20. B. Kiepuszewski, A. H. M. ter Hofstede, and Ch. Bussler. On Structured Workflow Modelling. In *CAiSE 2000*, volume 1789 of *LNCS*, pages 431–445. Springer, 2000.
21. C. Larman. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.
22. R. Laue and J. Mendling. Structuredness and its Significance for Correctness of Process Models. *Information Systems and e-Business Management*, 8(3):287–307, 2010.
23. D. Liu and M. Shen. Workflow Modeling for Virtual Processes: an Order-preserving Process-view Approach. *Information Systems Journal*, 28(6):505–532, 2003.
24. P. Massuthe, A. Serebrenik, N. Sidorova, and K. Wolf. Can I Find a Partner? Undecidability of Partner Existence for Open Nets. *Information Processing Letters*, 108(6):374–378, 2008.
25. R. M. McConnell and F. de Montgolfier. Linear-time Modular Decomposition of Directed Graphs. *Discrete Applied Mathematics*, 145(2):198–209, 2005.
26. J. Mendling and W. M. P. van der Aalst. Formalization and Verification of EPCs with OR-Joins Based on State and Context. In *CAiSE 2007*, volume 4495 of *LNCS*, pages 439–453, Trondheim, Norway, 2007. Springer.
27. J. Mendling, H. A. Reijers, and W. M. P. van der Aalst. Seven Process Modeling Guidelines (7PMG). *Information & Software Technology*, 52(2):127–136, 2010.

28. T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.
29. V. Pankratius and W. Stucky. A Formal Foundation for Workflow Composition, Workflow View Definition, and Workflow Normalization based on Petri Nets. In *APCCM*, pages 79–88, Darlinghurst, Australia, 2005. ACS, Inc.
30. D. L. Parnas. On the Criteria to be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15:1053–1058, December 1972.
31. A. Polyvyanyy, L. García-Bañuelos, and M. Dumas. Structuring Acyclic Process Models. In *BPM 2010*, volume 6336 of *LNCS*, pages 276–293. Springer, 2010.
32. A. Polyvyanyy, L. García-Bañuelos, and M. Dumas. Structuring Acyclic Process Models. *Information Systems*, 2011.
33. A. Polyvyanyy, S. Smirnov, and M. Weske. On Application of Structural Decomposition for Process Model Abstraction. In *BPSC 2009*, volume 147 of *LNI*, pages 110–122, Leipzig, 2009. GI.
34. A. Polyvyanyy, S. Smirnov, and M. Weske. The Triconnected Abstraction of Process Models. In *BPM 2009*, volume 5701 of *LNCS*, pages 229–244. Springer, 2009.
35. H. A. Reijers, J. Mendling, and R. M. Dijkman. On the Usefulness of Subprocesses in Business Process Models. BPM Center Report BPM-10-03, BPMcenter.org, 2010.
36. H. A. Reijers and I. T. P. Vanderfeesten. Cohesion and Coupling Metrics for Workflow Process Design. In *BPM 2004*, volume 3080 of *LNCS*, pages 290–305. Springer, 2004.
37. W. Sadiq and M. E. Orlowska. Analyzing Process Models Using Graph Reduction Techniques. *Information Systems Journal*, 25(2):117–134, 2000.
38. M. Schrefl and M. Stumptner. Behavior-Consistent Specialization of Object Life Cycles. *ACM Transactions on Software Engineering and Methodology*, 11(1):92–148, 2002.
39. S. Smirnov, R. Dijkman, J. Mendling, and M. Weske. Meronymy-based Aggregation of Activities in Business Process Models. In *ER 2010*, volume 6412 of *LNCS*, pages 1–14. Springer, 2010.
40. S. Smirnov, H. Reijers, Th. Nugteren, and M. Weske. Business Process Model Abstraction: Theory and Practice. Technical report, Hasso Plattner Institute, 2010. `http://bpt.hpi.uni-potsdam.de/pub/Public/SergeySmirnov/abstractionUseCases.pdf`.
41. S. Smirnov, H. A. Reijers, and M. Weske. A Semantic Approach for Business Process Model Abstraction. In *CAiSE 2011*, volume 6741 of *LNCS*, pages 497–511. Springer, 2011.
42. S. Smirnov, M. Weidlich, and J. Mendling. Business Process Model Abstraction Based on Behavioral Profiles. In *ICSOC 2010*, volume 6470 of *LNCS*, pages 1–16, 2010.
43. W. Stevens, G. Myers, and L. Constantine. *Structured Design*, pages 205–232. Yourdon Press, Upper Saddle River, NJ, USA, 1979.
44. M. Stoer and F. Wagner. A Simple Min-cut Algorithm. *Journal of the ACM*, 44:585–591, July 1997.
45. A. Streit, B. Pham, and R. Brown. Visualization Support for Managing Large Business Process Specifications. In *BPM 2005*, volume 3649 of *LNCS*, pages 205–219. Springer, 2005.
46. B. F. van Dongen, A. K. A. de Medeiros, and L. Wen. Process Mining: Overview and Outlook of Petri Net Discovery Algorithms. *Transactions Petri Nets and Other Models of Concurrency*, 5460:225–242, 2009.
47. J. Vanhatalo, H. Völzer, and J. Koehler. The Refined Process Structure Tree. In *BPM 2008*, volume 5240 of *LNCS*, pages 100–115. Springer, 2008.
48. M. Weidlich, A. Barros, J. Mendling, and M. Weske. Vertical Alignment of Process Models - How Can We Get There? In *BPMDS 2009*, volume 29 of *LNBIP*, pages 71–84. Springer, 2009.

49. M. Weidlich, J. Mendling, and M. Weske. Efficient consistency measurement based on behavioral profiles of process models. *IEEE Transactions on Software Engineering*, 37(3):410–429, 2011.
50. M. Weidlich, A. Polyvyanyy, J. Mendling, and M. Weske. Efficient Computation of Causal Behavioral Profiles Using Structural Decomposition. In *Petri Nets 2010*, volume 6128 of *LNCS*, pages 63–83. Springer, 2010.
51. M. Weidlich, S. Smirnov, Ch. Wiggert, and M. Weske. Flexab—flexible business process model abstraction. In *CAiSE Forum 2011*, London, United Kingdom, 2011. To appear.
52. M. Th. Wynn, H. M. W. Verbeek, W. M. P. van der Aalst, A. H. M. ter Hofstede, and D. Edmond. Reduction Rules for YAWL Workflows with Cancellation Regions and OR-joins. *Information & Software Technology*, 51(6):1010–1020, 2009.