A Flexible Approach for Abstracting and Personalizing Large Business Process Models

Jens Kolb
Ulm University,
Germany
jens.kolb@uni-ulm.de

Manfred Reichert
Ulm University,
Germany
manfred.reichert@uni-ulm.de

ABSTRACT

In process-aware information systems (PAISs), usually, different user groups have distinguished perspectives on the business processes supported and on related business data. Hence, personalized views and proper abstractions on these business processes are needed. However, existing PAISs do not provide adequate mechanisms for creating and visualizing process views and process model abstractions. Usually, process models are displayed to users in exactly the same way as originally modeled. This paper presents a flexible approach for creating personalized views based on parameterizable operations. Respective view creation operations can be flexibly composed to either hide non-relevant process information or to abstract it. Depending on the parameterization of the selected view creation operations, one obtains process views with more or less relaxed properties, e.g., regarding the degree of information loss or the soundness of the resulting model abstractions. Altogether, the realized view concept allows for a more flexible abstraction and visualization of large business process models satisfying the needs of different user groups.[1]

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: Computer-aided software engineering (CASE); H.1.2 [**User/Machine Systems**]: Human factors

General Terms

Management, Design, Human Factors

Keywords

Process Model Abstraction, Process View, View Update, Process Visualization, Human-oriented Business Process Management, Process Change

1. INTRODUCTION

Process-aware information systems (PAISs) provide support for business processes at the operational level [1]. A PAIS strictly separates process logic from application code, relying on explicit *process models*. This enables a *separation of concerns*, which is a well established principle in computer science to increase maintainability and to reduce costs of change [2]. The increasing adoption of PAISs has resulted in large process model collections (cf. Figure 1). In turn, each process model may refer to different domains, organizational units, and user groups, and comprise dozens or even hundreds of activities (i.e., process steps) [3]. Usually, different user groups need customized views on the process models relevant for them, enabling a personalized process model abstraction and visualization [4, 5, 6, 7]. For example, managers rather prefer an abstract process overview, whereas process participants need a more detailed view of the process parts they are involved in.

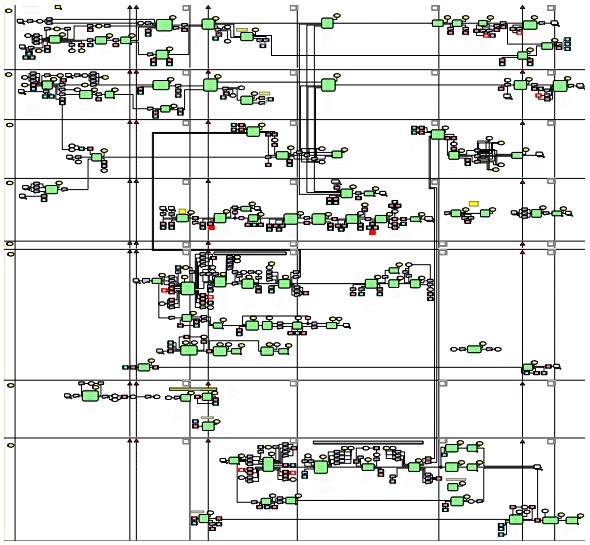

Figure 1. Complex Process Model (Partial View)

Hence, providing personalized process views is a much needed PAIS feature. Several approaches for creating process model abstractions based on process views have been proposed [8, 9]. However, none of them provide parametrizable operations to assist users in easily creating or changing process views. Furthermore, existing approaches do not consider another fundamental aspect of flexible PAISs: change and evolution [1, 2]. More precisely, it is not possible to change a large process model through editing or updating one of its view-based abstractions.

In the *proView*[2] project, we address these challenges in an integrated and consistent way by supporting the creation and visualization of process views as well as enabling users to change a process model through updates of a related process view. In this context, all other views associated with the changed process model need to be migrated to the new model version as well. Besides view-based abstractions and changes, *proView* allows for alternative process model appearances (e.g., tree-, form-, and diagram-based representation) as well as interaction techniques (e.g., gesture- vs. menu-based) [10, 11, 12, 13]. Note that the *proView* project extends previous results from our *Proviado* project [14, 15] by providing sophisticated change features and process visualizations. Our overall goal is to enable domain experts to "understand" and "interact" with the (executable) process models they are involved in.

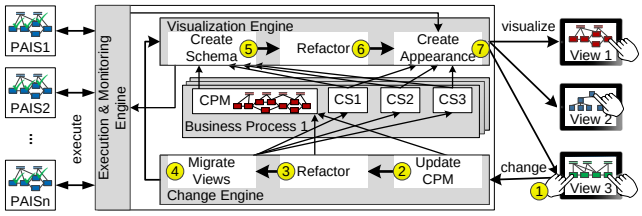

Figure 2. The proView Framework

Figure 2 gives an overview of the *proView* framework: A *business process* is captured and represented through a *Central Process Model (CPM)*. In addition, for a particular CPM, so-called *creation sets (CS)* are defined. Each creation set specifies the schema and appearance of a particular process view. For defining, visualizing, and updating process views, the *proView* framework provides engines for *visualization*, *change*, and *execution & monitoring*.

The *visualization engine* generates a process view based on a given CPM and the information maintained in a creation set CS, i.e., the CPM schema is transformed to the view schema by applying the corresponding *view creation operations* specified in CS (Step ⑤). Afterwards, the resulting view schema is *simplified* by applying well-defined *refactoring operations* (Step ⑥). Finally, Step ⑦ customizes the visual appearance of the view, e.g., creating a tree-, form-, or activity-based appearance [10, 14].

When a user updates a view schema, the *change engine* is triggered (Step ①-④), which updates the process schema of underlying CPM and updates all associated process views. [16] gives detailed insights into the *proView* architecture and the view update operations based on which business process models can be changed through updating process views.

This paper focuses on the parameterizable *visualization engine* component (cf. Figure 2), i.e., on the provision of a flexible and parameterizable component for creating process views and process model abstractions, respectively. Such a component must cover a variety of use cases. For example, it should be possible to create views only containing activities the current user is involved in or only showing non-completed process regions. As another example consider executable process models, which often contain technical activities (e.g., data transformation steps) to be excluded from visualization. Finally, selected process nodes may have to be hidden or aggregated to meet confidentiality needs [17].

[2]http://www.dbis.info/proView

The *proView* framework allows creating respective process views based on well-defined, parameterizable view operations. These rely on both graph reduction and graph aggregation techniques. While the former can be used to remove nodes from a process model, the latter are applied to abstract from certain process information (e.g., aggregating several activities to one abstract node). Additionally, *proView* supports the flexible composition of basic view operations to realize more sophisticated process model abstractions. The basic idea of creating process views has been already sketched in the context of Proviado [15, 18]. In this paper, we introduce more advanced view operations and their formal properties. Further, we outline their implementation. Finally, we combine elementary view creation operations enabling parameterizable high-level view operations.

Section 2 gives background information required to understand this paper. Section 3 introduces the formal foundations of parameterizable process views. Section 4 gives insights into practical issues and presents more complex examples for defining and creating process views. Section 5 presents the proof-of-concept prototype and a first validation we conducted. Section 6 discusses related work and Section 7 concludes with a summary.

2. BACKGROUNDS

Each process is represented by a process schema consisting of process nodes and the control flow between them (cf. Figure 3). For control flow modeling, control gateways (e.g., ANDsplit, XORsplit) and control edges are used.

Definition 1 (Process Schema): A *process schema* is defined by a tuple $P = (N, E, EC, NT, ET)$ where:

- N is a set of process nodes,
- $E \subset N \times N$ is a precedence relation (notation: $e = (n_{src}, n_{dest}) \in E$),
- $EC : E \rightarrow Conds \cup \{\text{TRUE}\}$ assigns optionally transition conditions to control edges,
- $NT : N \rightarrow \{Activity, ANDsplit, ANDjoin, ORsplit, ORjoin, XORsplit, XORjoin\}$ assigns to each $n \in N$ a node type $NT(n)$; N is divided into disjoint sets of activity nodes A ($NT = Activity$) and gateways S ($NT \neq Activity$).
- $ET : E \rightarrow \{ControlEdge, LoopEdge\}$ assigns a type $ET(e)$ to each edge $e \in E$.

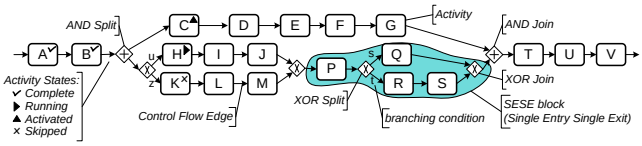

Figure 3. Example of a Process Instance

Note that this definition focuses on the control flow perspective. In particular, it can be applied to existing activity-oriented modeling languages (e.g., BPMN). Additional, view

creation operations specifically addressing the data flow are presented in [19]. Furthermore, the handling of loop structures is described [20].
We assume that a process schema has one start and one end node. Further, it has to be connected; i.e., each activity can be reached from the start node, and from each activity the end node is reachable. Finally, branches may be arbitrarily nested, but must be safe (e.g., a branch following a $XORsplit$ must not merge with an $ANDjoin$).

Definition 2 (SESE): Let $P = (N, E, EC, NT, ET)$ be a process schema and let $X \subseteq N$ be a subset of activity nodes. The subgraph P' induced by X is called *SESE* (Single Entry Single Exit) fragment iff P' is connected and has exactly one incoming and one outgoing edge connecting it with P. If P' has no preceding (succeeding) nodes, P' has only one outgoing (incoming) edge.

Based on a process schema P, related process instances can be created and executed at run-time. Regarding the process instance from Figure 3, for example, activities A and B are completed, C is activated (i.e., offered as work items in user worklists), H is running, and K is skipped (i.e., is not executed). Generally, a large number of process instances might run on a given process schema.

Definition 3 (Process Instance): A *process instance* I is defined by a tuple $(P, NS, \mathcal{H})$ where

- P denotes the process schema on which I is running,
- $NS : N \rightarrow ExecutionStates := \{NotActivated, Activated, Running, Skipped, Completed\}$ describes the execution state of each node $n \in N$,
- $\mathcal{H} = \langle e_1, \ldots, e_n \rangle$ denotes the execution history of I where each entry e_k is related either to the start or completion of a particular process activity.

For an activity $n \in N$ with $NS(n) \in \{Activated, Running\}$, all preceding activities either must be in state $Completed$ or $Skipped$, and all succeeding activities must be in state $NotActivated$. Further, there is a path π from the start node to n with $NS(n') = Completed \;\; \forall n' \in \pi$.

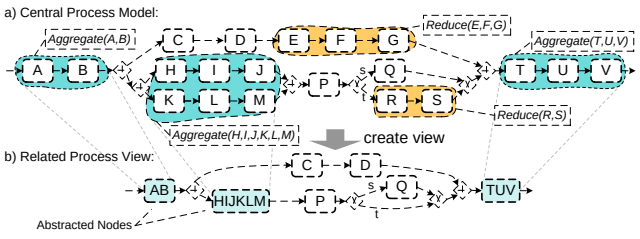

Figure 4. Example of a Process View

3. FUNDAMENTALS ON VIEW CREATION

We first introduce basic view creation operations and reason about the properties of the resulting process view schemas. As first example consider the process schema from Figure 4a. Assume that each of the activity sets $\{A, B\}$, $\{H, I, J, K, L, M\}$, and $\{T, U, V\}$ shall be aggregated, i.e., the process fragments induced by the respective activity set shall be replaced by one abstract activity. Further, assume that activity sets $\{E, F, G\}$ and $\{R, S\}$ shall be hidden from the user. Figure 4b shows a possible process view resulting from respective process model aggregations and reductions.
Generally, process views exhibit an information loss when compared to the original process (i.e., central process model (CPM)). As important requirement, view creation operations should have a precise semantics and be applicable to both process schemas and instances. Further, it should be possible to remove process nodes (i.e., *reduction*) or to replace them by abstracted ones (i.e., *aggregation*). When creating process views, it is fundamental to preserve the structure of non-affected process regions. Finally, the effects of view creation operations should be parameterizable to meet application needs best and to be able to control the degree of information loss in a flexible manner.
We first give an abstract definition of a *process view*. Note that the concrete properties of such a view depend on the view operations applied and their parameterization as specified in a respective creation set (CS).

Definition 4 (Process View): Let $P = (N, E, EC, NT, ET)$ be a process schema (i.e., central process model) with activity set $A \subseteq N$. Then: A *process view* on P is a process schema $V(P) = (N', E', EC', NT', ET')$ whose activity set $A' \subseteq N'$ can be derived from P by reducing and aggregating activities from $A \subseteq N$. Formally:

- $A_U = A \cap A'$ denotes the set of activities present in both P and $V(P)$,
- $A_D = A \setminus A'$ denotes the set of activities present in P, but not in $V(P)$; i.e., reduced or aggregated activities: $A_D \equiv AggrNodes \cup RedNodes$
- $A_N = A' \setminus A$ denotes the set of activities present in $V(P)$, but not in P.
  Each $a \in A_N$ is an abstract activity aggregating a set of activities from A:
  1. $\exists AggrNodes_i, i = 1, \ldots, n$ with
     $AggrNodes = \dot{\bigcup}_{i=1,\ldots,n} AggrNodes_i$
  2. There exists a bijective function $aggr$ with:
     $aggr : \{AggrNodes_i | i = 1, \ldots, n\} \rightarrow A_N$

Using the notions from Definition 4 for a given central process model P and related view $V(P)$, we introduce function $VNode : A \rightarrow A'$. This function maps each process activity $c \in A_U \cup AggrNodes$ to a corresponding activity in the respective process view:

$$VNode(c) = \begin{cases} c & c \in A_U \\ aggr(AggrNodes_i) & \exists i \in \{1, \ldots, n\}: \\ & c \in AggrNodes_i \\ undefined & c \notin A_U \cup AggrNodes \end{cases}$$

For each view activity $c' \in A'$, $VNode^{-1}(c')$ denotes the corresponding activity or the set of activities aggregated by c' in the central process model.
Finally, more complex process views are created by composing a set of view operations, which also define the semantics of the process view (cf. Section 4).

3.1 Creating Process Views Based on Schema Reduction

Any view management component should be able to remove activities in a process schema. For example, this is required to hide irrelevant or confidential process details from a particular user group. For this purpose, *proView* provides an elementary reduction operation (cf. Figure 5b). Based on it, higher-level reduction operations for hiding a set of activities are realized. Reduction of an activity (i.e., `RedActivity`) is realized by removing its node together with its incoming/outgoing edges from the process schema. Then, a new control edge is inserted between the predecessor and successor of the removed activity (cf. Figure 5b). For reducing activity sets, the single-aspect view operation REDUCECF is provided. Single-aspect operations focus on elements of one particular process aspect. Reduction is performed stepwise, i.e., for all activities to be removed, operation `RedActivity` is applied (cf. Figure 5a).

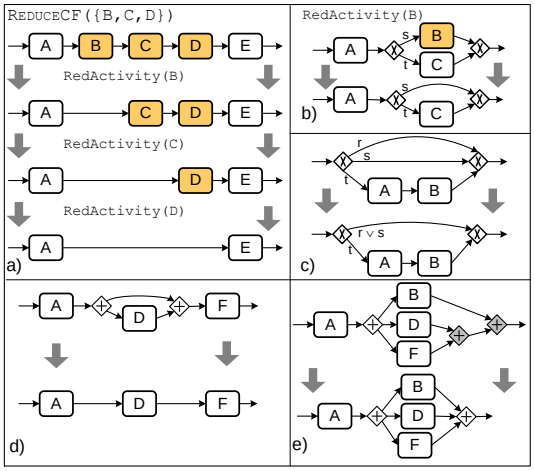

Figure 5. Reduction and Refactoring Operations

Note, the algorithms we apply are context-free and may introduce unnecessary process elements (e.g., empty branches). Respective elements are purged afterwards by applying well-defined, behaviour-preserving refactoring rules to the created view schema [21]. For example, when reducing a complete branch of a parallel branching, the resulting control edge may be removed as well (cf. Figure 5d). In case of an XOR-/OR-branching, however, the empty path (i.e., control edge) needs to be preserved. Otherwise an inconsistent schema would result (cf. Figure 5b). Similarly, when applying simplification rules to XOR-/OR-branches, respective transition conditions must be recalculated (cf. Figure 5c).
Note that any reduction of activities is accompanied by an information loss, while preserving the structure of the non-reduced schema parts, i.e., the activities present in both the process and the process view schema. The latter can be expressed using the notion of *order preservation*. For this, we introduce partial order relation $\preceq$ ($\subseteq N \times N$) on a process schema P with $n_1 \preceq n_2 \Leftrightarrow \exists$ path π in P from n_1 to n_2.

Definition 5 (Order-Preserving Views): Let $P = (N, E, EC, NT, ET)$ be a process schema with activity set $A \subseteq N$ and let $V(P) = (N', E', EC', NT', ET')$ be a view on P with activity set $A' \subseteq N'$. Then: $V(P)$ is called *order-preserving* iff $\forall n_1, n_2 \in A$ *with* $n_1 \neq n_2$ *and* $n_1 \preceq n_2 : n_1' = VNode(n_1) \wedge n_2' = VNode(n_2) \Rightarrow \neg(n_2' \preceq n_1')$.

This property expresses that the order of two activities in a process schema must not be reversed in a corresponding view. Obviously, the reduction operations depicted in Figure 5ab are order preserving. Generally, this property is fundamental for ensuring the integrity of process schemas and related view schemas. A stronger notion is provided by Definition 6. As we will see later, in comparison to Figure 5ab there are view operations which do not comply with Definition 6.

Definition 6 (Strong Order-Preserving Views): Let $P = (N, E, EC, NT, ET)$ be a process schema with activity set $A \subseteq N$ and let $V(P) = (N', E', EC', NT', ET')$ be a corresponding view with $A' \subseteq N'$. Then: $V(P)$ is *strong order-preserving* iff $\forall n_1, n_2 \in A$ *with* $n_1 \neq n_2$ *and* $n_1 \preceq n_2 : n_1' = VNode(n_1) \wedge n_2' = VNode(n_2) \Rightarrow n_1' \preceq n_2'$.

3.2 Creating Process Views Based on Schema Aggregation

The aggregation operation allows merging a set of activities into one abstracted activity. Depending on the structure of the subgraph induced by the respective activities, different schema transformations have to be applied. In particular, the aggregation of non-connected activities necessitates a more complex restructuring of the original process schema. Figure 6 shows the elementary operations provided for creating aggregated views. The depicted operations follow the policy to substitute the activities in-place by an abstract node (if possible), while ensuring properly order-preservation (cf. Definition 5). Note that *in-place substitution* is always possible when aggregating a SESE fragment (cf. Definition 2). If none of the operations from Figure 6ade can be applied, in turn, `AggrAddBranch` (cf. Figure 6b) is used. It identifies the nearest common ancestor and successor of all the activities to be aggregated and adds a new branch between them (cf. Figure 6b). Alternatively, aggregation of non-connected activities can be handled by applying elementary operations of type `AggrSESE` (cf. Section 4). Finally, when aggregating activities directly following a split node, there exist two alternatives (cf. Figure 6e): the first one aggregates activities applying `AggrAddBranch`, the second one shifts activities to the position preceding the split node (i.e., `AggrShiftOut`).
Except `AggrAddBranch`, the presented operations are strongly order-preserving (cf. Definition 6). However, `AggrAddBranch` violates this property. For example, in Figure 6b, order relation $D \preceq E$ can not be preserved when applying this operation.

Definition 7 (Dependency Set): Let $P = (N, E, EC, NT, ET)$ be a process schema with activity set $A \subseteq N$. Then: $\mathbb{D}_P = \{(n_1, n_2) \in A \times A | n_1 \preceq n_2\}$ is denoted as *dependency set* reflecting all direct and transitive control flow dependencies between any two activities.

We are interested in the relation between the dependency set of a process schema and a related view schema. For this purpose, let $\mathbb{D}_P$ be the dependency set of P and $\mathbb{D}_{V(P)}$ be the de-

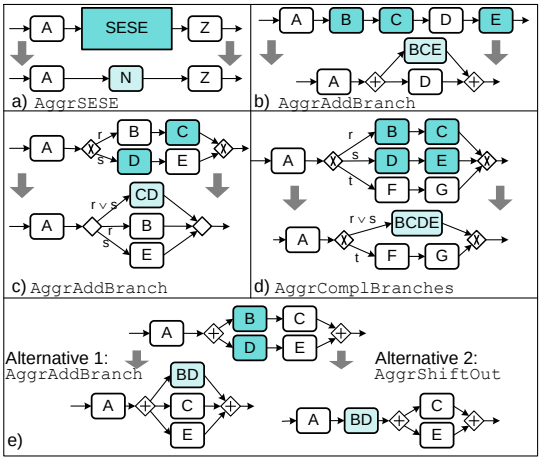

Figure 6. Elementary Aggregation Operations

pendency set of view $V(P)$. We further introduce a projection of the dependencies from $V(P)$ on P denoted as $\mathbb{D}'_{V(P)}$. The latter can be derived by substituting the dependencies of the abstract activities by the ones of the original activities. As example consider `AggrShiftOut` in Figure 6e. We obtain $\mathbb{D}_P$=$\{(A,B),(B,C),(C,F),(A,D),(D,E),(E,F)\}$ and $\mathbb{D}_{V(P)} = \{(A,BD),(BD,C),(BD,E),(C,F),(E,F)\}$. Further $\mathbb{D}'_{V(P)} = \{(A,B),(B,C),(D,C),(C,F),(A,D),(B,E),(D,E),(E,F)\}$ holds. As one can see, $\mathbb{D}'_{V(P)}$ contains additional dependencies. We denote this property as dependency-generating.

Calculation of $\mathbb{D}'_{V(P)}$: For $n_1 \in A_N$, $n_2 \in A'$: remove all $(n_1,n_2) \in \mathbb{D}_{V(P)}$; insert $\{(n,n_2)|n \in aggr^{-1}(n_1)\}$ instead (analogously for $n_2 \in A_N$); finally insert the dependencies between $AggrNodes$, i.e., $\mathbb{D}_P[AggrNodes] = \{d = (n_1,n_2) \in \mathbb{D}_P | n_1 \in AggrNodes \wedge n_2 \in AggrNodes\}$

Now we can classify effects on the dependencies between activities when building a view.

Definition 8 (Dependency Relations): Let $P = (N,E,EC,NT,ET)$ be a process schema and $V(P)$ be a corresponding view schema. Let further $\mathbb{D}_P$ and $\mathbb{D}'_{V(P)}$ be the dependency sets as defined above. Then:

- $V(P)$ is denoted as **dependency-erasing** iff there are dependency relations in $\mathbb{D}_P$ not existing in $\mathbb{D}'_{V(P)}$ anymore.
- $V(P)$ is denoted as **dependency-generating** iff $\mathbb{D}'_{V(P)}$ contains dependency relations not existing in $\mathbb{D}_P$.
- $V(P)$ is denoted as **dependency-preserving** iff it is neither dependency-erasing nor dependency-generating.

Generally, reduction operations are dependency-erasing. When aggregating activities, however, there exist elementary operations of all three types. In Figure 6, for example, `AggrSESE` is dependency-preserving, while `AggrAddBranch` is dependency-erasing since $(B,C) \notin \mathbb{D}'_{V(P)}$ holds. Finally, `AggrShiftOut` is dependency-generating: $(B,E) \in \mathbb{D}'_{V(P)}$. Theorem 1 expresses the relation between dependency properties (cf. Definition 8) and order-preservation property (cf. Definition 5).

Theorem 1: Let $P = (N,E,EC,NT,ET)$ be a process schema and let $V(P)$ be a corresponding view. Then:

i) $V(P)$ is dependency-erasing $\Rightarrow$
$V(P)$ is not strong order-preserving

ii) $V(P)$ is dependency-preserving $\Rightarrow$
$V(P)$ is strong order-preserving

The proof of Theorem 1 is based on the definition of the properties and dependency sets.

Proof: Let $P = (N,E,EC,NT,ET)$ be a process schema and $V(P)$ be a corresponding process view. Then:

i) $V(P)$ is dependency-erasing and relation $d = (a,b) \in \mathbb{D}_P$ exists with $d \notin \mathbb{D}'_{V(P)}$ (w.l.o.g., $a \in AggrNodes$ and $b \notin AggrNodes$). Hence, $a \not\preceq b$. Let $a' = VNode(a)$ and $b' = VNode(b)$, i.e., $a \in A'_N$ and $b \in A'_U$. Then $a' \not\preceq b'$ based on the calculation of $\mathbb{D}'_{V(P)}$. If $V(P)$ is strong order-preserving, dependency $a' \preceq b'$ relation has to exist.

ii) $V(P)$ is dependency-preserving, i.e., for all dependency relations $d = (a,b) \in \mathbb{D}_P : a \preceq b$. The calculation of $\mathbb{D}'_{V(P)}$ results in $a' \preceq b'$ with $a' = VNode(a)$ and $b' = VNode(b)$. The proof of Theorem 1 is based on the transitivity of the relation $\preceq$.

□

To conclude, we have presented a set of elementary aggregation operations. Each of them fits to a specific ordering of the activities to be aggregated. In Section 4.1 we combine these operations into more complex ones utilizing the discussed properties. Furthermore, this section has focused on the control flow schema of a process view. Generally, additional process aspects must be covered, including data elements, data flow, and process attributes (cf. Section 3.4) as defined for the different process elements (e.g., activities or data elements). When creating a process view, *proView* considers these aspects as well. Regarding data elements, for instance, Figure 7a shows an example of a simple aggregation; in this example, data edges connecting activities with data elements are re-linked when aggregating $\{B,C,D,E\}$ in order to preserve a valid model. More details about view operations covering the data flow perspective and corresponding correctness issues are discussed in [19].

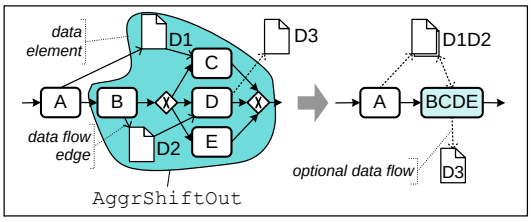

Figure 7. Aggregation of Data Elements

3.3 Applying Views to Process Instances

So far, we have only considered views on process schemas. This section additionally introduces views on process instances (cf. Definition 3). When creating respective instance views their execution state (i.e., states of concrete and abstract activities) must be determined and the trace relating to the instance view logically needs to be adapted. Examples are depicted in Figure 8a (reduction) and Figure 8b (aggregation). Function VNS then calculates the state of an abstract activity based on the states of the aggregated activities.

$$VNS(X) = \begin{cases} NotActivated & \forall x \in X: NS(x) \notin \{Activated, Running, Completed\} \wedge \\ & \exists x \in X: NS(x) = NotActivated \\ Activated & \exists x \in X: NS(x) = Activated \ \wedge \forall x \in X: NS(x) \notin \{Running, Completed\} \\ Running & \exists x \in X: NS(x) = Running \vee \\ & \exists x_1, x_2 \in X: NS(x_1) = Completed \wedge (NS(x_2) = NotActivated \vee NS(x_2) = Activated) \\ Completed & \forall x \in X: NS(x) \notin \{NotActivated, Running, Activated\} \wedge \\ & \exists x \in X: NS(x) = Completed \\ Skipped & \forall x \in X: NS(x) = Skipped \end{cases}$$

Definition 9 (View on Process Instance): Let $I = (P, NS, \mathcal{H})$ be an instance of schema $P = (N, E, EC, NT, ET)$ and $V(P) = (N', E', EC', NT', ET')$ be a view on P with corresponding aggregation and reduction sets $AggrNodes$ and $RedNodes$ (cf. Definition 4). Then: $V(I)$ on I is a tuple $(V(P), NS', \mathcal{H}')$ with:

- $NS' : N' \rightarrow ExecutionStates$ with $NS'(n') = VNS(VNode^{-1}(n'))$ assigns to each view activity a corresponding execution state.
- $\mathcal{H}'$ is the reduced/aggregated history of the instance. It is derived from $\mathcal{H}$ by (1) removing all entries e_i related to activities in $RedNodes$ and (2) for all j: replacing the first (last) occurrence of a start event (end event) of activities in $AggrNodes_j$ and remove the remaining start (end) events.

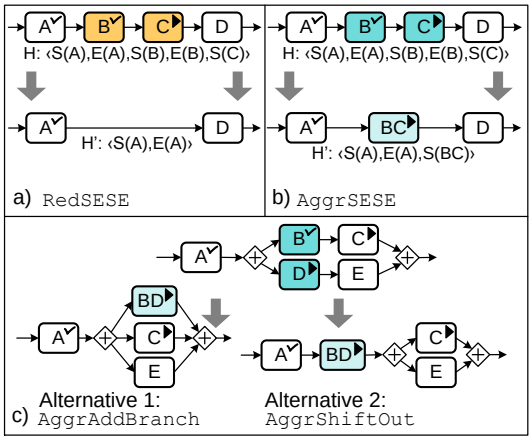

Figure 8. View Operations for Process Instances

Examples are depicted in Figure 8. Figure 8c shows the scenario from Figure 6e with execution states added. Note that applying `AggrShiftOut` yields an inconsistent state as two subsequent activities are in state *Running*.

Definition 10 (State Consistency): Let $I = (P, NS, \mathcal{H})$ be a process instance and let $V(I)$ be a corresponding view on I. Then:

- $V(I)$ is **strong state-consistent** iff for all paths π (cf. Section 2) in $V(I)$ from start to end, and not containing activities in state *Skipped*, there exists exactly one activity in state *Activated* or *Running*.
- $V(I)$ is **state-consistent** iff for all paths π in $V(I)$ from start to end and not containing activities in state *Skipped*, there does not exist more than one activity in state *Activated* or *Running*.

As indicated in Figure 8a, reducing activities from a process instance may result in a "gap" during instance execution, if no activity is in state *Running* or *Activated*. Hence, reduction is not strong state-consistent. Theorem 2 shows how state inconsistency is correlated with dependency relations (cf. Definition 8):

Theorem 2: Let I be a process instance and $V(I)$ a corresponding view. $V(I)$ is dependency-generating $\Rightarrow V(I)$ is not state-consistent.

Proof: Let $I = (P, NS, \mathcal{H})$ be a process instance of process $P = (N, E, EC, NT, ET)$ and $V(I) = (V(P), NS', \mathcal{H}')$ be a view on I with $V(P) = (N', E', EC', NT', ET')$ being dependency-generating. Then, there exists $n'_1, n'_2 \in N'$ in V which generates a dependency, i.e., $n'_1 \preceq n'_2$. Let $n_1 = VNode^{-1}(n'_1) \in N$ and $n_2 = VNode^{-1}(n'_2) \in N$. Then: $n_1 \not\preceq n_2$. Further there are two paths in P from start to end containing n_1 and n_2. Therefore, there exists a state of instance I with $NS(n_1) = Running$ and $NS(n_2) = Running$. Thus, $NS(n'_1) = Running$ and $NS(n'_2) = Running$. Since there is a path from start to end in $V(P)$, containing both n'_1 and n'_2, the claim follows. □

Theorem 2 shows that `AggrShiftOut` always causes an inconsistent execution state, whereas the application of operation `AggrAddBranch` maintains a consistent state.
Concerning the process instances, *proView* allows aggregating collections of them; i.e., multiple instances of the same process schema may be condensed to an aggregated one in order to provide abstracted information regarding their progress or key performance data.

3.4 View Operations Affecting Attributes of Process Elements

Process nodes are not elementary, but constitute complex objects comprising various *process attributes* (e.g., attributes of an activity may be cost, start time, and end time). In Figure 9, activities A, B, and C are aggregated into an abstracted activity ABC. When applying this aggregation, related attributes must be aggregated as well. For this purpose, *proView* provides *transformation functions*, which may be applied to aggregate attributes, e.g., function CONCAT allows concatenating the name of aggregated activities.

Table 1. Properties of View Creation Operations

Operation	Properties						
	str. order preserving	order preserving	str. state consistent	state consistent	depend. preserv.	depend. erasing	depend. generat.
RedActivity	+	+	-	+	-	+	-
AggrSESE	+	+	+	+	+	-	-
AggrComplBranches	+	+	+	+	+	-	-
AggrShiftOut	+	+	-	-	-	-	+
AggrAddBranch	-	+	+	+	-	+	-
AggrAttr	+	+	+	+	+	+	+

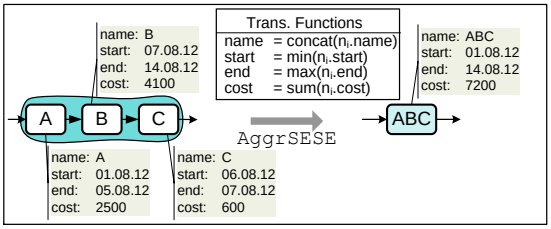

Figure 9. Aggregation of Attributes

Generally, there exist two application scenarios in which transformation functions are applied to process attributes:

- *AS1 (Integrated):* Transformation functions are integrated with view creation operations (as indicated in Figure 9). This scenario is particularly relevant for aggregation operations.
- *AS2 (Stand-Alone):* Transformation functions are applied to aggregate or reduce selected process attributes in the respective process view.

To discuss the application of transformation functions, Definition 1 has to be enriched with attributes.

Definition 11 (Process Schema with Attributes): A *process schema* is defined by a tuple $P = (N, E, EC, NT, ET, attr, val)$ with:

- N, E, EC, NT, ET as defined in Definition 1 and $\mathcal{A}$ the set of supported attributes.
- $attr : N \cup E \rightarrow AS$ assigns to each process element a corresponding attribute set $AS \subseteq \mathcal{A}$.
- $val : (N \cup E) \times AS \rightarrow valueDomain(AS)$ assigns to an attribute $a \in AS$ of process element $n \in (N \cup E)$ a respective value:

$$val(n,a) = \begin{cases} value\ of\ a, & a \in attr(n) \\ null, & a \notin attr(n) \end{cases}$$

Accordingly, a process view with attributes is denoted as $V(P) = (N', E', EC', NT', ET', attr', val')$. Further, $A.x$ denotes process attribute x of process node A.

In the context of a process schema, attributes may be grouped into four categories:

- *C1 (Activity State):* This category includes process attribute *VNS* representing the execution state of a process node (cf. Section 3.3).
- *C2 (Default Attributes):* This category represents default attributes that are common to all process nodes. For example, each process node comprises attributes describing its *name*, *start*, and *end time*.
- *C3 (Type-specific Attributes):* This category comprises all process attributes, which are only available for a specific type of process node (e.g., activity or data element). Attribute *cost*, for example, is only available for activities, but is undefined for gateways.
- *C4 (Other Attributes):* This category comprises all process attributes not available for all occurrences of a specific type of process node. For example, process attribute *personnel cost* may be only available for activities executed by a human resource.

In the following, view creation operations considering process attributes are presented:

Attribute Reduction Operation: View creation operation `ReduceAttr(A.x)` removes attribute $A.x$ in the corresponding process view. For example, in Figure 10 attribute $A.z$ is reduced when creating the respective process view. Generally, function `ReduceAttr(A.x)` may be applied to hide specific attributes from a user for privacy reasons [17].

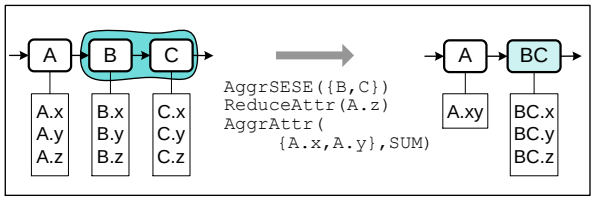

Figure 10. Combination of Attribute Operations

Attribute Aggregation Operation: View creation operation `AggrAttr(AS', func)` combines a set of process attributes `AS'` to an attribute using transformation function *func*.

Definition 12 (Transformation Function): Based on a set of attribute values, *transformation function* $func$ calculates a new attribute value. Let $\mathbb{W}_i$ with $i = 1, \ldots, n$ and $\mathbb{W}$ be the value domains of selected attributes as well as the abstracted attribute

$$func : \mathbb{W}_1 \ldots \mathbb{W}_n \rightarrow \mathbb{W}.$$

Depending on the type of attributes to be aggregated, various transformation functions can be applied. For example, a transformation function aggregating activity execution states (i.e., VNS) has been introduced in Section 3.3. Table 2 summarizes transformation functions supported by *proView*. For example, function `SUM` adds numerical values

of attributes returning their sum (cf. Figure 10).

Control Flow Aggregation Operations: As aforementioned, when aggregating activities (e.g., using AggrSESE), their attributes need to be aggregated as well (cf. Figure 10). For this purpose, transformation functions, as introduced in Definition 12, are applied. To automatically aggregate the attributes of different activities, *proView* provides default transformation functions. For example, attribute *start (end) time* may be aggregated using the MIN (MAX) transformation function (cf. Table 2). For aggregating activity *names*, function CONCAT can be applied. Generally, it is possible to override such standard behaviour in a given context.

Finally, when aggregating XOR/OR branches, we must take into account that not all branches will be executed in all cases. For example, summing up all available attribute values in such a situation might result in erroneous attribute aggregation. Regarding numerical transformation functions, execution probabilities of the different branches are used to calculate the attribute values expected. If such values are not available, one might initially assume that all branches are selected with the same probability. Note that for textual transformation functions, this is not required.

Table 2. Trans. Functions for Attribute Values

Transformation Functions for Numeric Values	
SUM	sum of attribute values
AVG	average of attribute values
MIN	minimum of attribute values
MAX	maximum of attribute values
COUNT	number of attributes
Transformation Functions for Textual Values	
CONCAT	concatenation of attribute values
FIRST	first value of attribute list
LAST	last value of attribute list
MAXFREQ	most frequently used attribute value
MINFREQ	less frequent used attribute value
RANDOM	random attribute value

4. ADVANCED VIEW CREATION CONCEPTS

To enable more complex view creation operations, the elementary operations presented in Section 3 may be combined. Table 1 summarizes the view properties we can guarantee for the elementary operations described. Based on this, we may reason about the properties of the views resulting from the combined use of elementary operations. For any process visualization component, however, manually selection of the elementary operations applied in the given context is inconvenient for users. Note that this would require in-depth knowledge of the different operations and their semantics. To tackle this challenge, *proView* additionally provides single-aspect view operations on top of elementary operations. In turn, this allows us to cope with more complex use cases as well. Single-aspect operations analyze the context of the activities to be reduced or aggregated in a process schema, and they automatically determine the appropriate elementary operations required to build the view with the desired properties.

4.1 Parameterizable View Creation Operations

One major use case of our process view framework is process visualization. However, other use cases (e.g., process modeling) are covered as well. Thus, different requirements regarding the properties of the resulting view schema exist. In one of our case studies, in the automotive domain, for example, we have shown that for the visualization of large process models minor inconsistencies or information loss will be tolerated by the users as long as an appropriate visualization can be obtained for them. Opposed to this, inconsistencies will not be accepted if process updates based on views shall be enabled [16].

To deal with these varying requirements, *proView* expands single-aspect operations with a parameter set. This allows specifying the properties of the resulting view schema. The parameters of operation AggregateCF, for example, are summarized in Table 3; e.g., when aggregating activities directly succeeding an ANDsplit (cf. Figure 6e) and requiring state-consistency of the resulting view `AggrAddBranch` has to be chosen (cf. Figure 11a).

In certain cases, the specified parameters might be too strict; i.e., no elementary view operations exist to realize the desired properties. We provide two strategies for addressing such scenarios. The first one subdivides the set of activities until elementary operations can be applied and the desired properties be ensured. The second strategy expands the activity set to achieve this. Regarding reduction, in turn, view generation is always possible due to the way activity sets are split into single activities and the application of `RedActivity`.

Figure 11 illustrates the use of the single-aspect operation AggregateCF. It depicts a process schema together with the set of activities to be aggregated. The view operation analyzes the structure of the activities and determines which elementary operation shall be applied. If the application of these operations results in a view schema complying with the properties defined by the desired parameters (*dependencies*, *execution states*), it is applied as shown in Figure 11a. If parameter *strategy* forces us to process the set of activities as it is, and an appropriate operation cannot be found, view generation is aborted with an error message. Figure 11b shows the result we obtain when expanding the activity set to be aggregated to the minimum SESE-block that contains all activities to be aggregated. Note that this strategy has

Table 3. Overview of Parameters for AggregateCF

Parameter	**Values**[1]	**Description**
dependencies	preserving, non-erasing, non-generating, **any**	The view operation applied should be dependency-preserving, not dependency-erasing, or not dependency-generating. Otherwise no restrictions regarding dependencies are considered.
exec. states	**inconsistent**, consistent	The view operation applied should be state consistent or may be state inconsistent.
strategy	**as-is**,subdivide, expand	Activity set should be aggregated as-is, may be subdivided or expanded.

[1] default values are printed in bold face

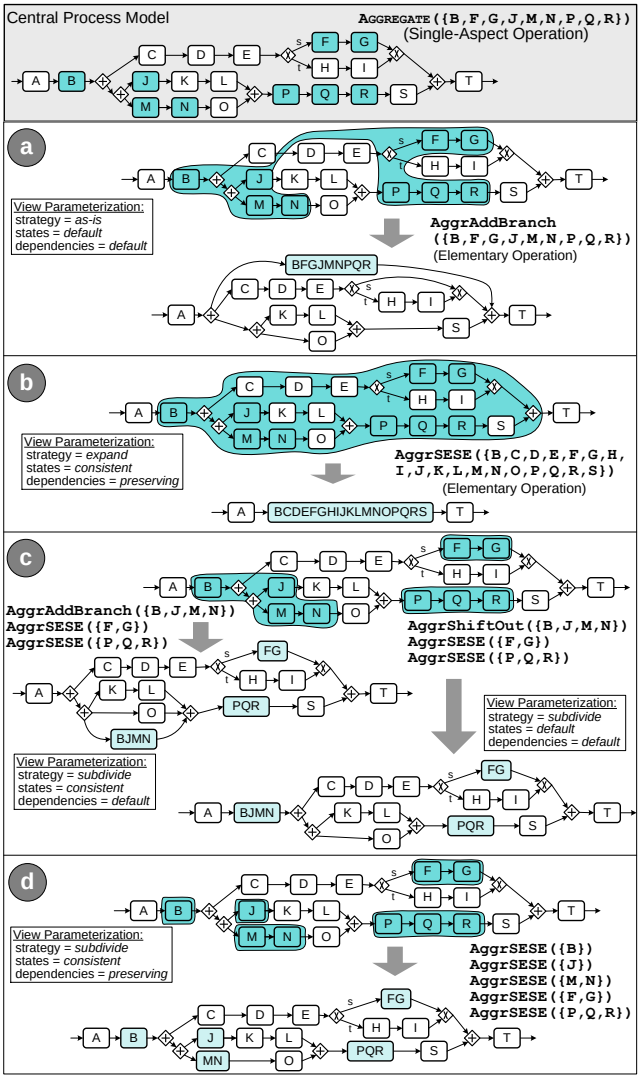

Figure 11. Views Depending on Quality Parameters

been proposed in literature as well [22, 23]. Generally, for visualization purposes it is not always acceptable to aggregate activities originally not contained in the aggregation set. Figure 11cd can be derived by subdividing the set of activities to be aggregated. This is done stepwise: First, all connected fragments are identified (cf. Figure 11c). If the aggregation of these fragments does not meet the required properties, the fragments are further subdivided until each subset constitutes a SESE (cf. Figure 11d).

Altogether, parameterization of view operations significantly increases the flexibility of our view creation approach. Further, it allows defining exactly the view properties to be preserved. Considering reduction, a parameterization at the level of single-aspect operations is not useful since reduction of a complex set of activities can be realized by calling `RedActivity` repeatedly as explained in Section 3.1.

4.2 A Leveled Operational Approach for Realizing Views

So far, we have presented a set of elementary and single-aspect view creation operations. Additionally, *proView* offers high-level operations hiding as much complexity from end-users as possible. As configuration parameter, the single-aspect view operations take the sets of activities to be reduced or aggregated, and then determine appropriate elementary operations. What is still needed are view operations allowing for a predicate-based specification of the respective activity sets. Besides, operations with *built-in intelligence* are useful, e.g. "show only the activities of a particular user role".

To meet these requirements, *proView* organizes view operations into four layers (cf. Figure 12). Thereby, high-level operations may access lower-level ones. For defining a view, operations from all layers can be used.

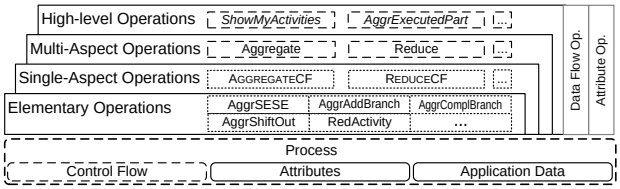

Figure 12. Multi-Layer View Operations

Elementary operations are designed for a specific ordering of the selected activity set within the process schema (cf. Section 3). *Single-aspect operations* receive a set of activities as input to be processed. They analyze the structure of the activities in the process schema and select the appropriate elementary operations based on the chosen parameterization. In turn, *multi-aspect operations* consider elements of different type (e.g., activities, data elements) and delegate their processing to single-aspect operations. *High-level operations* abstract from the aggregations or reductions neccessary to build a particular view: *AggrExecutedPart* only shows those parts of the process schema, that still may be executed, and aggregates already finished activities. Figure 13 shows an example illustrating the way high-level operation *AggrExecutedPart* is translated into a combination of single-aspect and elementary operations, respectively. In a first step, all activities with completed and skipped execution states are determined (i.e., activites A, B, C, E, H, and I). Based on this activity set *AggregateCF* with the following parameter settings is applied: `strategy=as-is, states=default`, and `dependencies=default` (cf. Section 4.1). Finally, in the resulting process view the execution states are set.

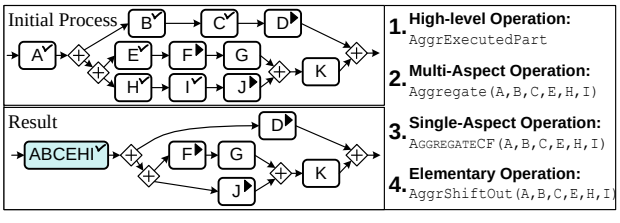

Figure 13. Example of AggrExecutedPart

Another high-level operation, *ShowMyActivities*, extracts exactly those parts of a process model the user is involved in.

Figure 14 shows an example of applying this operation to create a personalized process view for user *U1*. Operation *ShowMyActivities* first determines all activities the selected user *U1* is *not* involved in and reduces them. Finally, refactoring operations are applied to simplify the control-flow.

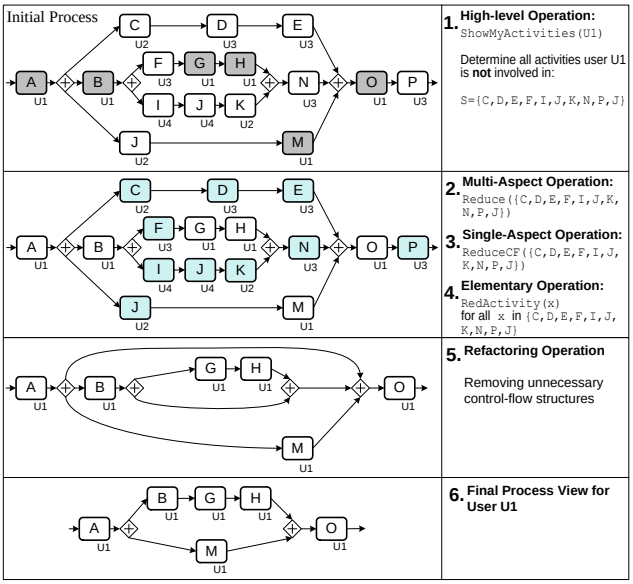

Figure 14. Example of ShowMyActivities

Note that *proView* supports additional operations at the different levels to cover data flow and attributes as well; e.g., to handle adjacent data elements when aggregating activities (remove, aggregate, or maintain) [16, 20].

5. EVALUATION

The *proView* framework presented in this paper has been implemented as a proof-of-concept prototype in a client-server application. This prototype enables users to simultaneously create and change process models based on process views [24, 25]. Overall, the *proView* prototype demonstrates the applicability of our framework (cf. Figure 15).

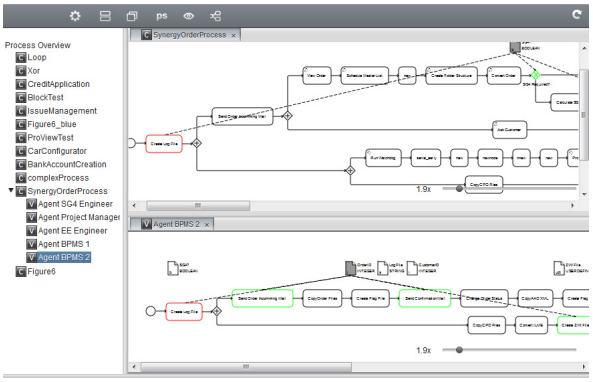

Figure 15. Proof-of-Concept Prototype

We further applied this prototype in an industry project, e.g., to visualize the *order handling process* of a mid-sized company for different user groups. This process consists of 56 activities and involves six different user roles. In the top right, Figure 15 shows this process; on the bottom right, an automatically generated view of an involved electrical-electronic engineer is displayed. This view is generated through high-level operation `ShowMyActivities`.

We automatically created views for each involved user measuring the complexity of the resulting view schemas based on well-known process metrics, i.e., number of activities, number of gateways, and McCabe metric measuring the complexity of the control flow schema [26, 27]. The results are depicted in Table 4. The first row shows the metrics for the initial process model *Order Process*. In the rows below, calculated metrics of the automatically generated views for user roles clerk, accountant, electrical-electronic engineer, mechanical engineer, and project manager are listed.

Table 4. Abstraction through High-Level Operation

Process	#Activities	#Gateways	McCabe
CPM: Order Process	56	14	8
V1: Clerk	2	0	0
V2: Accountant	17	2	1
V3: EE Engineer	17	2	2
V4: Mech. Engineer	11	0	0
V5: Proj. Mgmr.	9	2	1

As it can be seen, when providing personalized process views to users, process complexity can be reduced for them. In the given scenario, the number of activities is reduced to 2-17 depending on the respective view, i.e., to 3%-30% of the initial process model. Furthermore, the number of gateways decreases from initially 14 to 0-2 gateways, i.e., the resulting personalized process views have at most one branching (i.e., one split and join gateway). Finally, McCabe metric for control flow complexity is between 0 and 2 in the process views, which is a significant decrease. The calculated metrics show that process model size as well as complexity is decreased in the process views.

Overall, this evaluation has shown promising results. In particular, it becomes easier for process participants to understand those process aspects relevant for them.

6. RELATED WORK

IEEE 1471 recommends user-specific viewpoints for software architectures [28]. These viewpoints are templates from which individual views are created for a concrete software architecture. Since this standard does not define any methods, tools or processes, *proView* could provide a powerful framework in the context of PAIS. [29] introduces a meta model for views and shows a general overview of process view patterns. However, no implementation is provided.
Some view creation approaches deal with inter-organizational processes and apply views to create abstractions of private processes hiding sensitive process parts [30, 9, 31, 32]. In particular, views are specified by the designer.

[8] presents an approach with predefined view types (i.e. human tasks, collaboration view). As opposed to proView, it is limited to the specified view types and it is not possible to define user-specific views. [33] applies graph reduction to verify structural properties of process schemas. proView ac-

complishes this via aggregation and respective high-level operations. [34] uses SPQR-tree decomposition for abstracting process models. This approach neither provides high-level abstractions nor does it take other process aspects (e.g. data flow) into account.

[35] determines semantic similarity between activities by analysing the structural information of a process model. The discovered similarity is used to abstract the given process model. However, the approach neither distinguish between different user perspectives on a process model nor does it provide concepts to manually create process views.

An approach for creating aggregated views is presented in [36]. It proposes a two-phase procedure for aggregating parts of a process model that must not be shown to public. However, this approach focuses on block-structured graphs and neither considers data flow nor attributes are considered.
Implementations of process views focusing on process monitoring are presented in [37, 38]. These approaches focus on the mapping of run-time information to process views. Respective views have to be pre-specified manually by the designer.

Several approaches exist that align business process models with technical workflow models [39, 40, 41, 42] and to synchronize them with changes. However, neither an automated synchronization of changes nor high-level operations are provided. In this context, *proView* supports high-level operations to automatically create and update both business and technical process models [25].
An approach enabling abstractions of large, object-centric process structures is presented in [43]. In particular, state abstractions and coordination components are used to visualize (and execute) process structures.

The *proView* project provides a holistic framework for user-centric view creation based on elementary as well as high-level operations. Thereby, the behaviour and information perspectives are taken into account. Additionally, it considers run-time information. None of the existing approaches covers all these aspects. Furthermore, existing approaches for creating views are based on rigid constraints not taking practical requirements into account. For example, our first design of a visualization-oriented view mechanism was based on reduction and aggregation techniques for block-structured process graphs [20]. Presenting this solution to business users, however, we figured out that block-structured aggregation does not always meet the practical requirements coming with the visualization of large processes. For this reason, proView allows to flexibly specify the acceptable degree of imprecision. A validation of *proView* was conducted, where users are confronted with with complex, long-running development processes [20].

7. SUMMARY AND OUTLOOK

We introduced the *proView* framework and its formal foundation. Further, high-level view creation operations provide the required flexibility since process schemas can be adapted to specific user groups. Reduction operations provide techniques hiding irrelevant parts of the process, whereas aggregation operations allow abstracting from process details by aggregating arbitrary sets of activities in one node. Finally, parameterization of the respective operations allows specifying the quality level the resulting view schema must comply with. This enables adaptable process visualization not feasible with existing approaches. We have implemented large parts of the described view mechanism in a prototype and evaluated it in the context of an industry project. For usability reasons, it is important to provide appropriate methods for defining and maintaining process views. Therefore, we have designed and implemented a comprehensive set of user-oriented, high-level operations as well as on a view definition language. Both will be evaluated in a user experiment.

8. REFERENCES

[1] Reichert, M., Weber, B.: Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies. Springer (2012)

[2] Weber, B., Sadiq, S., Reichert, M.: Beyond Rigidity - Dynamic Process Lifecycle Support: A Survey on Dynamic Changes in Process-Aware Information Systems. Computer Science - Research and Development **23** (2009) 47–65

[3] Weber, B., Reichert, M., Mendling, J., Reijers, H.A.: Refactoring Large Process Model Repositories. Computers in Industry **62** (2011) 467–486

[4] Reichert, M.: Visualizing Large Business Process Models: Challenges, Techniques, Applications. In: 1st Int'l Workshop on Theory and Applications of Process Visualization (TAProViz'12), BPM'12 Workshops (Invited Keynote), Tallinn, Estonia (2012)

[5] Reichert, M., Kolb, J., Bobrik, R., Bauer, T.: Enabling Personalized Visualization of Large Business Processes through Parameterizable Views. In: Proc 26th Symposium On Applied Computing (SAC'12), Riva del Garda (Trento), Italy (2012) 1653–1660

[6] Streit, A., Pham, B., Brown, R.: Visualization Support for Managing Large Business Process Specifications. In: Proc 3rd Int'l Conf. Business Process Management (BPM'05). (2005) 205–219

[7] Kabicher-Fuchs, S., Rinderle-Ma, S., Recker, J., Indulska, M., Charoy, F., Christiaanse, R., Dunkl, R., Grambow, G., Kolb, J., Leopold, H., Mendling, J.: Human-Centric Process-Aware Information Systems (HC-PAIS). CoRR **abs/1211.4** (2012)

[8] Tran, H.: View-Based and Model-Driven Approach for Process-Driven, Service-Oriented Architectures. TU Wien, Dissertation (2009)

[9] Chiu, D.K., Cheung, S., Till, S., Karlapalem, K., Li, Q., Kafeza, E.: Workflow View Driven Cross-Organizational Interoperability in a Web Service Environment. Inf. Techn. and Mgmt. **5** (2004) 221–250

[10] Kolb, J., Reichert, M.: Using Concurrent Task Trees for Stakeholder-centered Modeling and Visualization of Business Processes. In: Proc S-BPM ONE 2012, CCIS 284. (2012) 237–251

[11] Kolb, J., Rudner, B., Reichert, M.: Towards Gesture-based Process Modeling on Multi-Touch Devices. In: Proc 1st Int'l Workshop on Human-Centric Process-Aware Information Systems (HC-PAIS'12), Gdansk, Poland (2012) 280–293

[12] Kolb, J., Hübner, P., Reichert, M.: Automatically Generating and Updating User Interface Components

in Process-Aware Information Systems. In: Proc 10th Int'l Conf on Cooperative Information Systems (CoopIS 2012). (2012) 444–454
[13] Kolb, J., Hübner, P., Reichert, M.: Model-Driven User Interface Generation and Adaptation in Process-Aware Information Systems. Technical report, UIB 2012-04, Technical Report, Ulm University (2012)
[14] Bobrik, R., Bauer, T., Reichert, M.: Proviado - Personalized and Configurable Visualizations of Business Processes. In: Proc 7th Int'l Conf Electronic Commerce & Web Technology (EC-WEB'06), Krakow, Poland (2006) 61–71
[15] Bobrik, R., Reichert, M., Bauer, T.: View-Based Process Visualization. In: Proc 5th Int'l Conf. on Business Process Management, Brisbane, Australia (2007) 88–95
[16] Kolb, J., Kammerer, K., Reichert, M.: Updatable Process Views for User-centered Adaption of Large Process Models. In: Proc 10th Intl. Conf. on Service Oriented Comp. (ICSOC'12), Shanghai, China (2012)
[17] Reichert, M., Bassil, S., Bobrik, R., Bauer, T.: The Proviado Access Control Model for Business Process Monitoring Components. Enterprise Modelling and Inf. Sys. Arch. - An Int'l Journal **5** (2010) 64–88
[18] Rinderle, S., Bobrik, R., Reichert, M., Bauer, T.: Business Process Visualization - Use Cases, Challenges, Solutions. In: Proc 8th Int'l Conf on Enterprise Information Systems (ICEIS'06). Number May, Paphos, Cyprus (2006) 204–211
[19] Kolb, J., Reichert, M.: Data Flow Abstractions and Adaptations through Updatable Process Views. In: Proc 27th Symposium On Applied Computing (SAC'13), Coimbra, Portugal (2013) (to appear)
[20] Bobrik, R.: Konfigurierbare Visualisierung komplexer Prozessmodelle. Phd thesis, Ulm University (2008)
[21] Weber, B., Reichert, M.: Refactoring Process Models in Large Process Repositories. In: Proc 20th CAiSE, Montpellier, France (2008) 124–139
[22] Liu, D.R., Shen, M.: Workflow Modeling for Virtual Processes: an Order-Preserving Process-View Approach. Information Systems **28** (2003) 505–532
[23] Shen, M., Liu, D.R.: Discovering Role-Relevant Process-Views for Recommending Workflow Information. In: Proc 14th DEXA'03, Prague, Czech Republic (2003) 836–845
[24] Kolb, J., Kammerer, K., Reichert, M.: Updatable Process Views for Adapting Large Process Models: The proView Demonstrator. In: Proc of the Business Process Management 2012 Demonstration Track, Tallinn, Estonia (2012)
[25] Kolb, J., Reichert, M.: Supporting Business and IT through Updatable Process Views: The proView Demonstrator. In: Demo Track of the 10th Int'l Conference on Service Oriented Computing (ICSOC'12), Shanghai, China (2013) 460–464
[26] Cardoso, J., Mendling, J., Neumann, G., Reijers, H.A.: A Discourse on Complexity of Process Models. In: Proc Workshops 4th Int'l Conf on Business Process Management (BPM'06), Vienna, Austria (2006) 117–128
[27] Laue, R., Gruhn, V.: Complexity Metrics for Business Process Models. In: Proc 9th Int'l Conference on Business Information Systems (BIS'2006), Klagenfurt, Austria (2006) 1–12
[28] IEEE: (IEEE 1471-2000 - Recommended Practice for Architectural Description for Software-Intensive Systems)
[29] Schumm, D., Leymann, F., Streule, A.: Process Viewing Patterns. In: Proc 14th IEEE International EDOC Conference, EDOC 2010, IEEE Computer Society (2010) 89–98
[30] Chebbi, I., Dustdar, S., Tata, S.: The View-based Approach to Dynamic Inter-Organizational Workflow Cooperation. Data & Know. Eng. **56** (2006) 139–173
[31] Kafeza, E., Chiu, D.K.W., Kafeza, I.: View-Based Contracts in an E-Service Cross-Organizational Workflow Environment. In: Techn. E-Services. (2001) 74–88
[32] Schulz, K.A., Orlowska, M.E.: Facilitating Cross-Organisational Workflows with a Workflow View Approach. Data & Knowledge Engineering **51** (2004) 109–147
[33] Sadiq, W., Orlowska, M.E.: Analyzing Process Models Using Graph Reduction Techniques. Information systems **25** (2000) 117–134
[34] Polyvyanyy, A., Smirnov, S., Weske, M.: The Triconnected Abstraction of Process Models. In: Proc 7th Int'l Conf. Business Process Management. (2009)
[35] Smirnov, S., Reijers, H.A., Weske, M.: A Semantic Approach for Business Process Model Abstraction. In: Advanced Information Systems Engineering, Springer Berlin / Heidelberg (2011) 497–511
[36] Eshuis, R., Grefen, P.: Constructing Customized Process Views. Data & Know. Engineering **64** (2008)
[37] Shan, Z., Yang, Y., Li, Q., Luo, Y., Peng, Z.: A Light-Weighted Approach to Workflow View. APWeb 2006 (2006) 1059–1070
[38] Schumm, D., Latuske, G., Leymann, F., Mietzner, R., Scheibler, T.: State Propagation for Business Process Monitoring on Different Levels of Abstraction. In: Proc. 19th ECIS, Helsinki, Finland (2011)
[39] Weidlich, M., Barros, A., Mendling, J., Weske, M.: Vertical Alignment of Process Models - How Can We Get There? In: Proc Ent., Bus. Proc. & Inf. Sys. Mod. Volume 29 of Lecture Notes in Business Information Processing. Springer (2009) 71–84
[40] Branco, M.C., Troya, J., Czarnecki, K., Küster, J., Völzer, H.: Matching Business Process Workflows Across Abstraction Levels. In: Proc 15th Int'l Conf Model Driven Engineering Languages and Systems (MODELS'12), Innsbruck, Italy (2012)
[41] Favre, C., Küster, J., Völzer, H.: The Shared Process Model. In: Demo Track 10th Int'l Conf on Business Process Management (BPM'12). (2012) 12–16
[42] Buchwald, S., Bauer, T., Reichert, M.: Bridging the Gap Between Business Process Models and Service Composition Specifications. In: Service Life Cycle Tools and Technologies: Methods, Trends and Advances. IGI Global (2011) 124–153
[43] Künzle, V., Reichert, M.: PHILharmonicFlows: Towards a Framework for Object-Aware Process Management. Journal of Software Maintenance and Evolution: Research and Practice **23** (2011) 205–244