# Parameterizable Views for Process Visualization*

Ralph Bobrik[1], Manfred Reichert[2], and Thomas Bauer[3]

[1] Institute of Databases and Information Systems, Ulm University, Germany,
ralph.bobrik@uni-ulm.de
[2] Information Systems Group, University of Twente, The Netherlands,
m.u.reichert@cs.utwente.nl
[3] DaimlerChrysler Group Research & Adv. Engineering, GR/EPD, Ulm, Germany,
thomas.tb.bauer@daimlerchrysler.com

**Abstract.** In large organizations different users or user groups usually have distinguished perspectives over business processes and related data. Personalized views on the managed processes are therefore needed. Existing BPM tools, however, do not provide adequate mechanisms for building and visualizing such views. Very often processes are displayed to users in the same way as drawn by the process designer. To tackle this inflexibility this paper presents an advanced approach for creating personalized process views based on well-defined, parameterizable view operations. Respective operations can be flexibly composed in order to reduce or aggregate process information in the desired way. Depending on the chosen parameterization of the applied view operations, in addition, different "quality levels" with more or less relaxed properties can be obtained for the resulting process views (e.g., regarding the correctness of the created process view scheme). This allows us to consider the specific needs of the different applications utilizing process views (e.g., process monitoring tools or process editors). Altogether, the realized view concept contributes to better deal with complex, long-running business processes with hundreds up to thousands of activities.

## 1 Introduction

In order to streamline their way of doing business, companies have to deal with a large number of processes involving different domains, organizations, and tasks. Often, these business processes are long-running, comprise a large number of activities, and involve a multitude of user groups. Each of these user groups needs a different view on the process with an adapted visualization and a customized granularity of information [1]. For example, managers usually prefer an abstract overview of the process, whereas process participants need a more detailed view on the process parts they are involved in. In such scenarios, personalized process visualization is a much needed functionality. Despite its practical importance, current BPM tools do not offer adequate visualization support. Very often, processes are displayed to the user in more or less the same way as drawn by the

* This work has been funded by *DaimlerChrysler Group Research*

designer. There are some tools which allow to alter the graphical appearance of a process and to hide selected process aspects (e.g., data flow). Sophisticated concepts for building and managing process views, however, are missing.

In the Proviado project we are developing a generic approach for visualizing large processes consisting of hundreds up to thousands of activities. To elaborate basic visualization requirements we conducted several case studies in the automotive domain [2]. This has led us to three dimensions needed for process visualizations [3]. First, it must be possible to reduce complexity by discarding or aggregating process information not relevant in the given context. Second, the notation and graphical appearance of process elements (e.g., activities, data objects, control connectors) must be customizable. Third, different presentation forms (e.g., process graph, swim lane, calendar, table) should be supported.

This paper focuses on the first dimension, i.e., the provision of a flexible component for building process views. Such a view component must cover a variety of use cases. For example, it must be possible to create process views which only contain activities the current user is involved in or which only show non-completed process parts. Often, process models contain technical activities (e.g., data transformation steps) which shall be excluded from visualization. Finally, selected process elements may have to be hidden or aggregated to meet confidentiality constraints. To enable such use cases, the Proviado approach allows to create process views based on well-defined, parameterizable view operations. Basically, we distinguish between two kinds of view operations either based on graph reduction or on graph aggregation techniques. While the former can be used to remove elements from a process graph, the latter are applied to abstract process information (e.g., by aggregating a set of activities to an abstract one). Proviado allows to flexibly compose basic view operations in order to reduce or aggregate process information in the desired way.

The realization of view operations is by far not trivial, and may require complex process transformations. Regarding activity aggregations, for example, they may be more or less complex depending on the set of activities to be aggregated. In particular, the aggregation of not connected activities constitutes a big challenge, which requires advanced aggregation techniques. Furthermore, it is of fundamental importance that view properties (e.g., regarding the created process view scheme) can be flexibly set depending on application needs. If process visualization is in the focus, for example, these properties may be relaxed. Opposed to this, more rigid constraints with respect to process views become necessary if these views shall be updatable. In Proviado we achieve this flexibility by the support of parameterized view operations.

The remainder of this paper is structured as follows: Section 2 gives background information needed for the understanding of this paper. In Section 3 we introduce our theoretical framework for realizing parameterizable and configurable process views. Section 4 gives insights into practical issues and presents a more complex example for defining and creating process views. Section 5 discusses related work. The paper concludes with a summary and an outlook in Section 6.

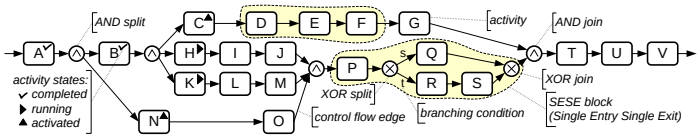


**Fig. 1.** Example for a process instance

## 2 Backgrounds

We introduce backgrounds needed for the understanding of the paper. In a process-aware information system each business process is represented by a process scheme $P$; i.e., a process graph which consists of (atomic) activities and control dependencies between them (cf. Fig. 1). For control flow modeling we use control edge as well as structural activities (e.g., ANDsplit, XORsplit).

**Definition 1.** *A process scheme is a tuple* $P = (N, E, EC, NT)$ *where*

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

This definition only covers the control flow perspective. However, the view operations presented in Sec. 3 consider other process aspects (e.g., data elements and data flow) as well. Though loop backs are supported by our framework we exclude them here; i.e., we only consider acyclic process graphs. Further, process schemes have to meet several constraints: First, we assume that a process scheme has one start and one end node. Second, each process scheme has to be connected; i.e., each activity can be reached from the start node, and from each activity the end node is reachable. Third, branchings may be arbitrarily nested. Fourth, a particular path in a branching must be merged with an eligible join node (e.g., a branch following an XORsplit must not merge with an ANDjoin).

**Definition 2 (SESE).** *Let* $P = (N, E, EC, NT)$ *be a process scheme and let* $X \subseteq N$ *be a set of activity nodes. The subgraph* $P'$ *induced by* $X$ *is called* SESE *(Single Entry Single Exit) iff* $P'$ *is connected and has exactly one incoming edge and one outgoing edge connecting it with* $P$*.*

At run-time new process instances can be created and executed based on a process scheme $P$. Regarding the process instance from Fig. 1, for example, activities $A$ and $B$ are completed, activities $C$ and $N$ are activated (i.e., offered in user worklists), and activities $H$ and $K$ are running.

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

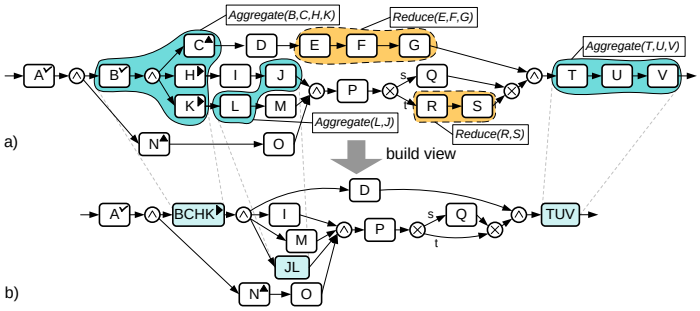


**Fig. 2.** Example for a process view

- *$P$ denotes the process scheme on which $I$ is based*
- *$NS : N \rightarrow ExecutionStates := \{NotActivated, Activated, Running, Skipped, Completed\}$ describes the execution state for 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.*

We informally summarize selected properties regarding the consistency of instance states. For an activity $n \in N$ with $NS(n) \in \{Activated, Running\}$ all activities preceding $n$ either must be in state $Completed$ or in state $Skipped$, and all activities succeeding $n$ must be in state $NotActivated$. Furthermore, there is a path $\pi$ from the start node to $n$ with $NS(n') = Completed \quad \forall n' \in \pi$.

## 3 View Fundamentals

We introduce basic view operations and argue about properties of resulting process views. As a first example consider the process instance from Fig. 2a. Assume that each of the activity sets $\{B, C, H, K\}$, $\{J, L\}$, and $\{T, U, V\}$ shall be aggregated, i.e., each of them shall be replaced by one abstract node. Assume further that activity sets $\{E, F, G\}$ and $\{R, S\}$ shall be "removed" from this process instance. A possible view resulting from respective aggregations and reductions is depicted in Fig. 2b. Generally, process views exhibit an information loss when compared to the original process. As an important requirement view operations should have a precise semantics and be applicable to both process schemes and process instances. Furthermore, it must be possible to remove process elements (*reduction*) or to replace them by abstracted elements (*aggregation*). When building process views it is also very important to preserve the structure of non-affected process parts. Finally, view building operations must be highly parameterizable in order to meet application requirements best and to be able to control the degree of desirable or tolerable information loss.

We first give an abstract definition of *process view*. The concrete properties of such a view depend on the applied view operations and their parameterization.

**Definition 4 (Process View).** *Let $P = (N, E, EC, NT)$ be a process scheme with activity set $A \subseteq N$. Then: A process view on $P$ is a process scheme $V(P) = (N', E', EC', NT')$ whose activity set $A' \subseteq N'$ has been derived from $P$ by reducing and/or aggregating activities from $A \subseteq N$. Formally:*

- *$A_U = A \cap A'$ denotes the set of activities present in $P$ and $V(P)$*
- *$A_D = A \setminus A'$ denotes the set of activities present in $P$, but not in $V(P)$; either these are reduced or aggr. 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:*

$$aggr : \{AggrNodes_1, \ldots, AggrNodes_n\} \rightarrow A_N$$

For a given process scheme $P$ with corresponding view $V(P)$ we introduce function $VNode : A \rightarrow A'$ based on the notions from Def. 4. $VNode(n)$ maps each process activity $n \in A_U \cup AggrNodes$ to a corresponding view activity:

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

Further, for each activity $c' \in A'$, $VNode^{-1}(c')$ denotes the corresponding activity in the original scheme or the set of activities aggregated by $c'$.

In Proviado, a process view is created by composing a set of view building operations: $V(P) = Op_n \circ \ldots \circ Op_1(P)$. The semantics of the generated view is then determined by these operations. Proviado uses a multi-layered operational approach (cf. Fig. 8). There are elementary view operations of which each is describing how a given set of activities has to be processed (e.g., reduce or aggregate) considering a particular process structure (e.g., sequence of activities, branching). On top of this, basic view operations are provided which analyze the structure of the activity set and then decide which elementary operations have to be applied.

### 3.1 Views Based on Graph Reduction

One basic requirement for a view component constitutes its ability to remove process elements (e.g., activities) if desired. For this purpose, Proviado provides a set of elementary reduction operations as depicted in Fig. 3. Based on these elementary operations, higher-level reduction operations (e.g., for removing an arbitrary set of activities) can be realized. Usually, reduction is applied if irrelevant information has to be removed from a process scheme or confidential process details shall be hidden from a particular user group.

In the following we characterize the view operations based on different formal properties. Reduction of activities always comes along with a loss of information

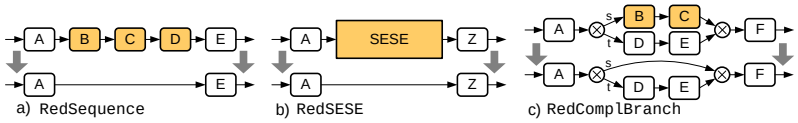


**Fig. 3.** Different elementary reduction operations

while preserving the overall structure of the remaining activities (i.e., the activities being present in the original process scheme as well as the view scheme). The latter property can be expressed taking the notion of *order preservation*. For this purpose we first introduce a partial order relation $\preceq$ ($\subseteq N \times N$) on the process scheme $P$ with $n_1 \preceq n_2 :\Leftrightarrow \exists$ path in $P$ from $n_1$ to $n_2$.

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

This property reflects the natural requirement that the order of two activities in a process scheme must not be reversed when building a corresponding view. Obviously, the elementary reduction operations depicted in Fig. 3 are order preserving. More generally, this property is basic for the integrity of process schemes and corresponding views. A stronger notion is provided by Def. 6. This stronger property is also met by reduced views. As we see later, however, there are view operations which do not comply with the property specified by Def. 6.

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

We first look at elementary reduction operations (cf. Fig. 3). The reduction of an activity sequence (`RedSequence`) is realized by removing the respective activities from the process scheme and by adding a new control edge instead. Reduction of a SESE block (cf. Fig. 3b) is performed similar to `RedSequence`. Since there are only two edges connecting the block with its surrounding, the SESE block is completely removed and a new edge between its predecessor and successor is added. In order to reduce not connected activity sets the basic view operation REDUCE is provided. Reduction is performed stepwise. First, the activities to be reduced are divided into subsets, such that each sub-graph induced by a respective subset is connected. Second, to each connected sub-graph, the operation `RedSESE` is applied; i.e., REDUCE is based on the reduction of connected components using the elementary operation `RedSESE`. The algorithms we apply in this context are generic, and can therefore result in unnecessary process elements, e.g., non- needed structure nodes, empty paths in a parallel branching, etc. Respective elements are removed in Proviado afterwards by applying well-defined graph simplification rules to the created view scheme. Note that this is

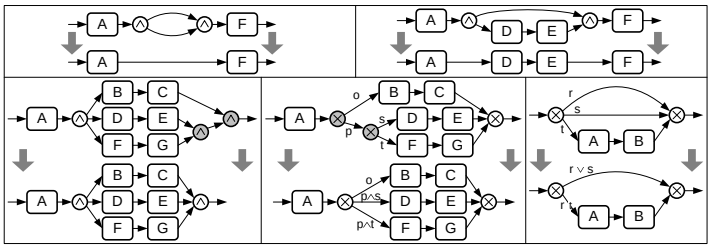


**Fig. 4.** Examples of Process Graph Simplification Rules

not always straightforward. For example, when reducing a complete branch of a parallel branching, the resulting control edge can be removed as well. As opposed to this, in case of an XOR- or OR-branching the empty path (i.e., control edge) must be preserved. Otherwise an inconsistent process scheme would result (cf. Fig. 3c). Similarly, when applying simplification rules to XOR- or OR-branchings the respective conditions $EC$ must be recalculated as depicted in Fig. 4.

### 3.2 Views Based on Graph Aggregation

As opposed to reduction-based views the aggregation operation aims at summarizing several activities into one abstracted activity. Depending on the concrete structure of the sub-graph induced by the set of activities to be aggregated, different graph transformations may have to be applied. In particular the aggregation of not connected activity sets necessitates a more complex restructuring of the process graph. Fig 5a-c show the elementary operations for building aggregated views. Elementary operations for activity sets contained within branching structures are shown in Fig. 5f-g. All these elementary operations follow the same policy to substitute the activities in-place with the abstract one if possible and maintaining the order-preservation. Note that in-place substitution is always possible when aggregation a SESE block. Otherwise a new branch is added (with additional split/join nodes if needed; cf. Fig 5c). Alternatively, the aggregation of not connected activities, as illustrated in Fig. 5d, can be handled by two separate elementary operations of type `AggrSequence`. Note that this alternative is handled by more complex (i.e. higher level) operations as will be explained in Section 4.

When aggregating activities directly following/preceding a split/join node there exist two alternatives for aggregating the activities (cf. Fig. 5e). Alternative 1 aggregates the activities applying `AggrAddBranch`, alternative 2 shifts the activities in front of the split node (`AggrShiftOut`). We discuss differing properties of the resulting views later.

All presented operations except `AggrAddBranch` are strong order-preserving. `AggrAddBranch` violates this property as, for example, the order relation $D \preceq E$ is not maintained when applying this operation (cf. Fig. 5c).

Before further investigating properties of aggregation operations we need the following definition.

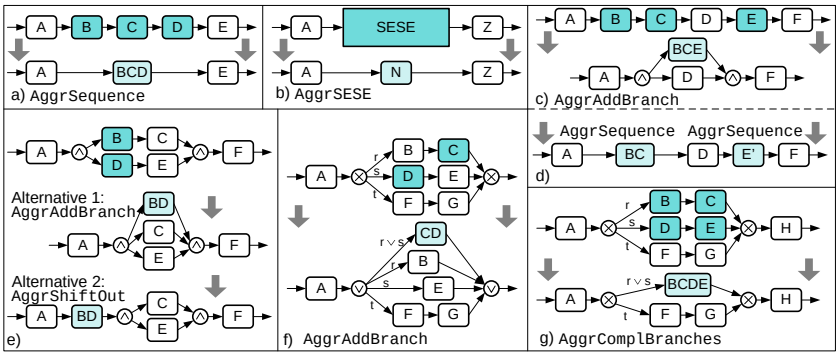


**Fig. 5.** Different elementary aggregation operations

**Definition 7 (Dependency set).** *Let $P = (N, E, EC, NT)$ be a process scheme with activity set $A \subseteq N$. Let further $\mathbb{D}_P \subseteq A \times A$. Then $\mathbb{D}_P$ is denoted as dependency set. It contains all direct and indirect control flow dependencies between activities.*

$$\mathbb{D}_P = \{(n_1, n_2) \in A \times A | n_1 \preceq n_2\}$$

We are interested in the relation between the dependency set of a process and a related process view. For this purpose, let $\mathbb{D}_P$ be the dependency set of $P$ and $\mathbb{D}_{V(P)}$ the dependency set of view $V(P)$ on $P$. We further need a projection of the dependencies from $V(P)$ on $P$ denoted by $\mathbb{D}'_{V(P)}$. It can be derived by substituting the dependencies of the abstract activities by the dependencies of the original activities. As example consider `AggrShiftOut` in Fig. 5e. 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. $\mathbb{D}_P[AggrNodes] = \{d = (n_1, n_2) \in \mathbb{D}_P | n_1 \in AggrNodes \wedge n_2 \in AggrNodes\}$

We now can classify exactly what happens to the dependencies between activities when building a view.

**Definition 8 (Dependency relations).** *Let $P = (N, E, EC, NT)$ be a process scheme and let $V(P)$ be a corresponding view. Let $\mathbb{D}_P$ and $\mathbb{D}'_{V(P)}$ be the dependency sets as defined above.*

- *$V(P)$ is denoted as **dependency erasing** iff there exist 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 contained in $\mathbb{D}_P$.*
- *$V(P)$ is denoted as **dependency preserving** iff it is neither dependency erasing nor dependency generating.*

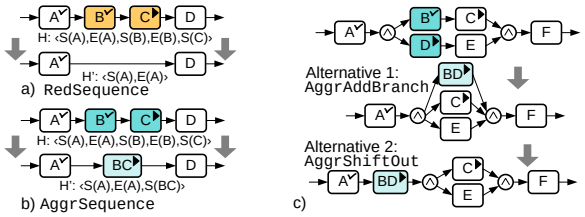


**Fig. 6.** View operations for process instances

Taking this definition it should be clear that every reduction operation is dependency erasing. When aggregating activities there exist elementary operations of all three types. In Fig. 5, for instance, `AggrSESE` is dependency preserving; by contrast `AggrAddBranch` is dependency erasing since $(B, C) \notin \mathbb{D}'_{V(P)}$; finally, `AggrShiftOut` is dependency generating: $(B, E) \in \mathbb{D}'_{V(P)}$.

Theorem 1 explains the relation of dependency properties (cf. Def. 8) and order-preservation (cf. Def. 5).

**Theorem 1 (Dependencies and order preservation).** *Let $P = (N, E, EC, NT)$ be a process scheme and $V(P)$ a corresponding view. Then:*

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

The proof of Theorem 1 follows directly from the definition of the properties and the dependency sets and is omitted here. In summary, we have presented a set of elementary aggregation operations. Each of them fits for a specific constellation of the activities to be aggregated. In Sec. 4.2 we describe how these operations can be combined to more complex operations taking advantage of the discussed properties.

### 3.3 Applying Views to Process Instances

So far we have only considered views for process schemes. In this section we additionally deal with view on process instances (cf. Def. 3). When building respective instance views the execution state of the created view, i.e. states of concrete and abstract activities, must be determined and the "history" for the instance view has to be adapted. Examples are depicted in Fig. 6a (reduction) and Fig. 6b (aggregation). The following function $VNS$ calculates the state of an abstract activity based on the states of the underlying 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 a Process Instance).** *Let $I = (P, NS, \mathcal{H})$ be an instance of process scheme $P = (N, E, EC, NT)$. Let further $V(P) = (N', E', EC', NT')$ be a view on $P$ with $AggrNodes$ and $RedNodes$ as corresponding aggregation and reduction sets (cf. Def. 4).*

*Then: A view $V(I)$ 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.*

Examples of instance views with corresponding history are depicted in Fig. 6a-b.

Fig. 6c shows the example from Fig. 5e with execution states added. Interestingly, applying `AggrShiftOut` yields an inconsistent state as two subsequent activities are in state *Running*. For characterizing this situation consider Def. 10.

**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 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 in $V(I)$ from start to end and not containing activities in state Skipped, there exists not more than one activity in state Activated or Running*

Theorem 2 shows how state inconsistency is correlated with the degree of preserved of dependencies.

**Theorem 2 (State consistency).** *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 $P = (N, E, EC, NT)$ and $V(I) = (V(P), NS', \mathcal{H}')$ a view on $I$ with $V(P) = (N', E', EC', NT')$ dependency-generating. Then there exist $n'_1, n'_2 \in N'$ in between of which $V$ generated 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 holds $n_1 \not\preceq n_2$. Further there are two paths in $P$ from

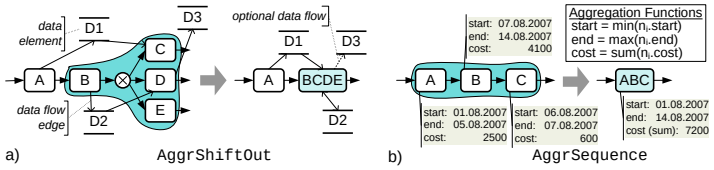


**Fig. 7.** Aggregation of (a) data elements and (b) attributes

start to end through $n_1$ and $n_2$. Therefore there exists a state of the base process instances $I$ with $NS(n_1) = Running$ and $NS(n_2) = Running$. Thus also $NS(n'_1) = Running$ and $NS(n'_2) = Running$. Since there is a path from start to end in $V(P)$, that contains both $n'_1$ and $n'_2$, follows the claim. □

This theorem shows that an aggregation using `AggrShiftOut` always provokes an inconsistent execution state while applying `AggrAddBranch` maintains a consistent state.

Regarding the aggregation of process instances in Proviado we also deal with the aggregation of collections of instances. Here multiple instances of the same process scheme are condensed to an aggregated one in order to provide abstracted information about business progress or to allow the calculation of key performance indicators.

### 3.4 Further Issues

Due to space limitation this paper focuses on control flow views. Generally, additional process aspects have to be covered including process data elements, data flow, and the many attributes defined for the different process objects. When building a process view, Proviado considers and processes these aspects as well. Regarding data elements, for instance, Fig. 7a shows an example of a simple aggregation. Here the data edges connecting activities with data elements must be re-routed when aggregating $\{B, C, D, E\}$ in order to preserve a valid model. Similarly, when performing an aggregation operation activity attributes must be processed as well by applied aggregation functions. Proviado offers a variety of attribute aggregation functions in this context. In Fig. 7b, for example, we applied a MIN-function to the aggregation set for determining start time, a MAX-function for calculating end time, and a SUM-function for aggregating cost attributes.

### 3.5 Discussion

The elementary operations presented so far allow to deal with all possible constellations of activities to be reduced or aggregated. Regarding reduction the completeness of the set of provided elementary operations is guaranteed by the way not connected activity sets are split into connected subsets and by applying

`RedSESE` to the connected sub-graphs. Similarly, `AggrAddBranch` ensures completeness for aggregation. In Proviado basic view operations for dealing with the different reduction and aggregation scenarios are provided on top of the stated elementary operations. In particular, these basic operations analyze the positions of the selected activities in a process scheme and determine the appropriate elementary operation to build the view.

Table 1 gives an overview of the properties that can be guaranteed for the different elementary operations. Based on this we can also argue about view properties when applying a set of elementary operations. Further, in Proviado we use these properties for parameterizing view operations, i.e. basic operations may be expanded by a set of properties the resulting view schema. For instance, when aggregating activities directly succeeding an AND-split (cf. Fig.5e) and the resulting view has to be state-consistent, Alternative 1 (i.e. `AggrAddBranch`) will be applied. Altogether the parameterization of view operations increases the flexibility of our view concept and allows defining exactly what properties must be preserved by view generation.

Of course, the parameters specified may be too strict, i.e. no elementary operation can be found guaranteeing the compliance with desired properties Sec. 4 introduces mechanisms for dealing with such scenarios.

## 4 View Application

So far we have presented a set of elementary and basic operations for manipulating a process schema when building a view. In practice, the user needs higher-level operations hiding the complexity of view generation as far as possible. The sketched basic view operations take as parameter a set of activities to be reduced or aggregated and then determine the appropriate elementary operations to be applied. What is still needed are view operations allowing for a predicate-based specification of the relevant set of activities. Even operations with more inherent intelligence are required, like "show only activities of a certain user".

In order to meet these requirements Proviado organizes view building operations in five layers (cf. Fig. 8). Doing so, operations at more elevated layers may access all underlying operations. For defining a view the user or application can make use of all operations provided in the different layers:

**Table 1.** Overview of elementary operations properties

| Operation | Properties | | | | | | |
|---|---|---|---|---|---|---|---|
| | str. order preserving | order preserving | str. state consistent | state consistent | depend. preserv. | depend. erasing | depend. generat. |
| `RedSequence` | + | + | - | + | - | + | - |
| `RedSESE` | + | + | - | + | - | + | - |
| `AggrSequence` | + | + | + | + | + | - | - |
| `AggrSESE` | + | + | + | + | + | - | - |
| `AggrComplBranches` | + | + | + | + | + | - | - |
| `AggrShiftOut` | + | + | - | - | - | - | + |
| `AggrAddBranch` | - | + | + | + | - | + | - |

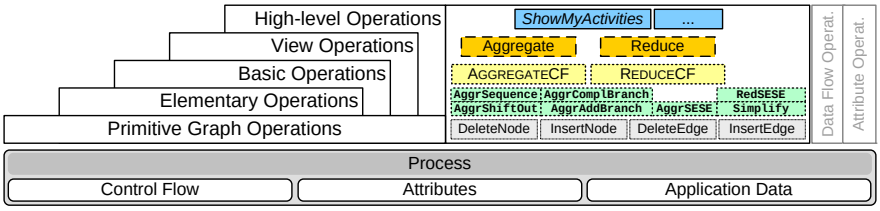


**Fig. 8.** Organization of view operations

*Primitive graph operations* are used for inserting and deleting nodes and edges at the process graph level. These operations build the bridge to the internal representation of process schemes.

*Elementary operations* are tailored for a specific constellation of selected activity set within the process scheme (cf. Sec. 3).

*Basic operations* receive a set of activities as input that has to be processed. They analyze the structure of the activities inside the process scheme and select the appropriate elementary operation.

*View operations* receive a predicate to determine the selected activity set. At this level there exists exactly one view operation for aggregation and reduction of control flow elements. The differentiation of layers *basic* and *view* becomes clear when considering operations for data flow and attributes (cf. 3.4).

*High-level operations* are independent from aggregation or reduction. They combine lower-level reduction and aggregation operations to realize more sophisticated functionalities. Currently we support selected operations. One of them extracts exactly those parts of the process the user is involved in (*ShowMyActivities*). The *UnfinishedProcessPart* operation only shows that part of the process which still has to be executed, and reduces or aggregates the already finished activities. The *Range* operation retrieves the subprocess between two activities. The set of high-level operations is intended to be extensible for operations added later. Below we give an example illustrating the way high-level operations are translated into a combination of basic, respectively elementary operations.

It should be noted here, that Proviado supports additional operations at the different levels for also manipulating data flow and for calculating of attribute values of aggregated activities. All these operations are triggered by the control flow operations at basic and elementary level.

### 4.1 Translating User-defined Views into View Building Operations

This section provides an example of a view based on a high-level operation and a predicate (cf. Fig. 9). The view shall generate a reduced process scheme that contains only activities the current user is involved in. For this purpose a high-level operation *ShowMyActivities* generates a predicate for a given user name. This predicate identifies all activities executed without participation of the user (Step 1; note the negation of the before mentioned set). Step 2 invokes

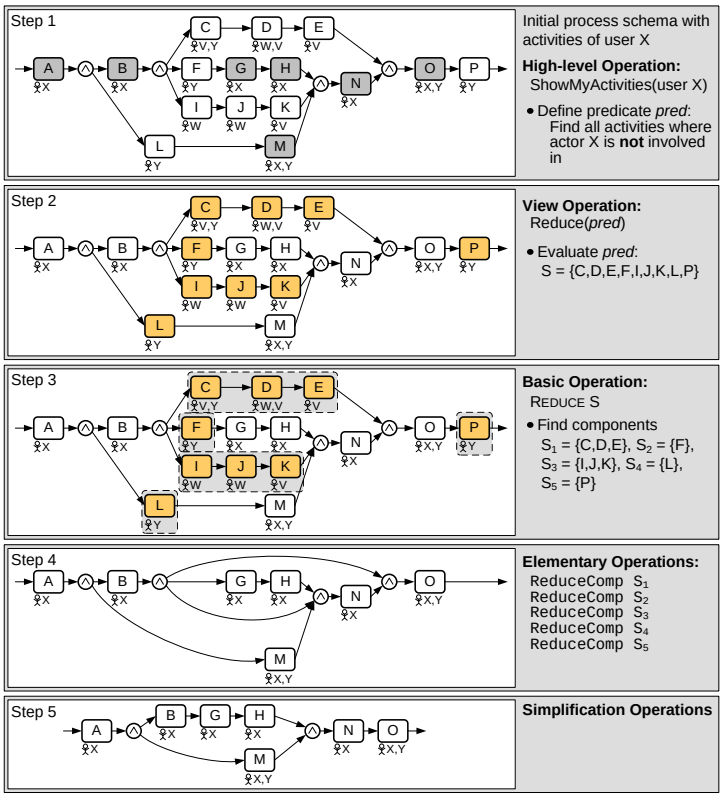


**Fig. 9.** Performing a complex view-operation

the view operation Reduce that calculates the set of relevant activities $S$. The basis operation REDUCECF then subdivides $S$ into connected components $S_i$ (Step 3). These connected components are then processed by the elementary operation `RedSESE` (step 4). Applying the appropriate simplification rules yields the compactified process scheme (Step 5). Obviously, the creation of a view results in a schema for which a new graph layout has to be recalculated. This is by far not trivial. Though there exist graph layout algorithms for aligning control flow graphs, there is still some need to deal with complex process graphs (including data elements, data flow, etc.).

### 4.2 Parameterization of View Building Operations

The primary use case of our view approach is the visualization of complex business processes. However, other applications can be supported as well. Thus different requirements regarding the consistency of the resulting process scheme exist. In our experience, for visualization purposes minor inconsistencies will acceptable if a better presentation can be obtained. Obviously, such inconsis-

tencies are not acceptable in case certain manipulations shall be applied to the views (e.g. view updates). To comply with these sometimes contradictory claims Proviado allows for the parameterization of view building operations. We already mentioned this concept in Sec. 3.2 where different elementary operations for aggregation are elected depending on parameters (cf. Fig. 5e).

Similarly, these parameters can influence view generation at a higher level. Think of a view operation AggregateCF{B,C,H,K,T,U,V} to be applied to the process scheme of Fig. 1. A direct translation into basic and elementary operations then leads to invocation of `AggrAddBranch({B,C,H,K,T,U,V})`. This inserts a new branch between the split after $A$ and the end. Parameterization allows to execute AggregateCF with the additional claim to preserve dependencies. Then three possibilities exist: (1) by subdividing the activities set into connected subsets, the view operation invokes basic operation AGGREGATE two times. Finally, `AggrShiftOut({B,C,H,K})` and `AggrShiftOut({T,U,V})` is called resulting in a state inconsistent view model. (2) If state inconsistency is forbidden by a parameter, the set $\{B, C, H, K, T, U, V\}$ is expanded until forming a SESE. Finally, `AggrSESE({B,...,V})` produces a high-quality but quite small process. (3) An additional parameter inhibits the extension of the node set. In this case a view generation is not possible revealing an error message to the user.

Considering reduction, a parameterization is useless at the level of view operations since reduction of a complex set of activities can be realized by calling `RedSESE` repeatedly. By contrast, for high-level operations a parameter that inhibits loss of information can be set to force view generation by aggregation.

## 5 Related Work

Process views have been an emerging topic in research for the last years. The majority of approaches focuses on inter-organizational processes. Here process views are used to build an abstracted version (public process) of an internal process (private process) in order to hide implementation details [4,5,6,7]. In [8,9] a top-down approach is used. Starting with a global process describing the interactions between partners, each participant details his part of the process inserting activities. For the insertion 'inheritance preserving transformation rules' are defined. All these approaches do not follow an operational approach, i.e. the process view are modeled by the process designer.

Regarding user-specific views, [10,11] provide techniques to extract the subgraph induced by the activities conducted by the user. We also have presented a high-level operation that accomplishes this task by applying reduction operations. Only few papers propose operations similar to reduction and aggregation. [12] uses graph reduction to verify the structural correctness of process schemes. Control flow structures are removed stepwise following certain rules. If no structural flaws were contained in the original model, this leads to a trivial graph.

View generation by aggregation is described in [13]. So called "Virtual" processes are built by aggregating connected sets of activities. [13] introduces a notion of order-preserving that corresponds to strong order-preserving as defined

in this paper. This property is mandatory and, consequently, an aggregation of not connected activity sets is not possible in this approach. Furthermore, view building by reduction is not provided. In total, this approach constitutes the most comprehensive view building mechanism. Even loops are discussed in detail in conjunction with order preservation. [14] extends the approach with a state mapping function allowing views on process instances. An implementation of a view model for monitoring purposes is presented in [15]. The view component can be plugged to existing workflow management systems using their APIs. [15] focuses on the mapping between original instance and process view at run time. Respective views are pre-modeled manually.

## 6 Summary and Outlook

For the personalized visualization of large, long-running processes with hundreds up to thousands activities the process schemes must be customizable retaining only the information relevant to the current user. In this paper we have introduced the theoretical framework of the Proviado view mechanism and have given an idea of the flexibility introduced by high-level operations in combination with parameterization. With this mechanism an adaptation of process schemes to specific user groups becomes feasible for the first time. Reduction operations provide techniques to hide irrelevant parts of the process, while aggregation operations allow for abstracting implementation details by summarizing arbitrary sets of activities in one activity. Additionally, parameterization of the operations allow for specifying the quality level to comply with.

We have been working on the implementation of the view mechanism and realized large parts of the concept. This mechanism is one component of the Proviado framework for flexible and user-specific visualization of business processes [3,16]. The framework shall be transferred to industrial practice in the context of business process monitoring. Thus, from the very beginning we have been interested in covering practical issues as discussed in Sec. 4 as well. There are still some topics left to be worked on. For usability reasons it is important to provide an acceptable way to define and maintain process view definitions. For this purpose, besides a comprehensive set of user-oriented high-level operations, we are thinking about a view definition language and/or a GUI. A future research topic of great interest for the BPM community is the development of sophisticated graph layout algorithms capable of handling complex process graphs.

## References


1. Streit, A., Pham, B., Brown, R.: Visualization support for managing large business process specifications. In: Proc. BPM '05. Volume 3649 of LNCS. (2005) 205–219
2. Bobrik, R., Reichert, M., Bauer, T.: Requirements for the visualization of system-spanning business processes. In: Proc. DEXA Workshops. (2005) 948–954
3. Bobrik, R., Bauer, T., Reichert, M.: Proviado - personalized and configurable visualizations of business processes. In: Proc. EC-Web '06. LNCS 4082 (2006)

4. Chebbi, I., Dustdar, S., Tataa, S.: The view-based approach to dynamic inter-organizational workflow cooperation. Data Knowl. Engin. **56**(2) (2006) 139–173
5. Chiu, D.K.W., Cheung, S.C., Till, S., Karlapalem, K., Li, Q., Kafeza, E.: Workflow view driven cross-organizational interoperability in a web service environment. Information Technology and Management **5**(3-4) (2004) 221–250
6. Kafeza, E., Chiu, D.K.W., Kafeza, I.: View-based contracts in an e-service cross-organizational workflow environment. In: Techn. for E-Services (TES '01). (2001)
7. Schulz, K.A., Orlowska, M.E.: Facilitating cross-organisational workflows with a workflow view approach. Data Knowledge Engineering **51**(1) (2004) 109–147
8. Aalst, W.v., Weske, M.: The P2P approach to interorganizational workflows. In: Proc. Conf. on Adv. Inf. Sys. Engineering (CAiSE). Volume 2068 of LNCS. (2001)
9. Aalst, W.v.: Inheritance of interorganizational workflows: How to agree to disagree without loosing control? Inf. Techn. and Mgmt Journal **2**(3) (2002) 195–231
10. Avrilionis, D., Cunin, P.Y.: Using views to maintain petri-net-based process models. In: Proc. of Int. Conf. on Software Maintenance (ICSM '95). (1995) 318–326
11. Shen, M., Liu, D.R.: Discovering role-relevant process-views for recommending workflow information. In: DEXA'03. Volume 2736 of LNCS. (2003) 836–845
12. Sadiq, W., Orlowska, M.E.: Analyzing process models using graph reduction techniques. Information Systems **25**(2) (2000) 117–134
13. Liu, D.R., Shen, M.: Workflow modeling for virtual processes: an order-preserving process-view approach. Information Systems **28**(6) (2003) 505–532
14. Liu, D.R., Shen, M.: Business-to-business workflow interoperation based on process-views. Journal of Decision Support Systems **38**(3) (2004) 399–419
15. Shan, Z., Yang, Y., Li, Q., Luo, Y., Peng, Z.: A light-weighted approach to workflow view implementation. In: Proc. APWeb. Volume 3841 of LNCS. (2006)
16. Bobrik, R., Bauer, T.: Towards configurable process visualizations with proviado. In: WS on Agile Coop. Proc.-Aware Inf. Sys. (ProGility'07). (2007) to appear.