Constructing customized process views

Constructing Customized Process Views

Rik Eshuis and Paul Grefen

Eindhoven University of Technology, Department of Technology Management
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{h.eshuis,p.w.p.j.grefen}@tm.tue.nl

Abstract. To enable effective cross-organizational collaborations, process providers have to offer views on their internal processes to consumers. A process view hides details of an internal process that are secret to or irrelevant for the consumer. This paper describes a formal two-step approach for constructing customized process views on structured process models. First, a non-customized process view is constructed from an internal structured process model by aggregating internal activities the provider wishes to hide. Second, a customized process view is constructed by aggregating and omitting activities from the non-customized view that are not requested by the consumer. To show the feasibility of the approach, an existing architecture for setting up dynamic virtual enterprises is extended.

1 Introduction

In today's networked society, more and more companies collaborate with each other in a virtual way through the internet. Each partner in such a collaboration network has its own private business process. Coordination of these local business processes takes place through the network. However, process-oriented collaboration in a network can only occur if partners reveal some details of their business processes to the network. These revealed details can be captured in an external or public view on the private business process [12]. A public view is like a window through which the network can monitor the operation of an underlying private business process of a partner.

Now, for each partner providing a process to the network, there is a trade off in the level of detail shown in the process view. On the one hand, if the process view reveals too few details of the underlying private business process, then the other partners of the network cannot effectively detect the local state of the business process, which might prevent an effective operation of the network. For example, if some partner responsible for producing a customer made component does not reveal any details of its internal production process, the main contractor cannot monitor the progress at the partner's site and therefore cannot coordinate the overal networked production process in a proper way. On the other hand, if the process view reveals all private details, including business secrets, then the provider runs the risk of loosing its competitive edge. Other partners might copy then its way of working, turning from collaborators into competitors.

Moreover, other partners in the network might not be interested in every detail of the private provider process. Some details may be irrelevant for them or simply be noise in the business relationship. In the example of the production network given above, the main contractor is likely not interested in detailed information about the production process, but only in high-level status information. Thus, process views need to be customized to the needs of the consumers.

This paper proposes a formal approach to construct a customized process view on a business process. The approach consists of two main phases. In the first phase, process providers construct process views that are independent of any specific consumer. These non-customized process views contain aggregate activities that hide activities of the internal business process. This way, a process view hides private details from the underlying process model. In the second phase, consumers tailor these process views to their needs. The resulting customized view reveal only those activities requested by the consumer; the remaining activities of the underlying process view are hidden or omitted. This way, a customized process view filters noise from the underlying process view. Though the first phase needs input from the provider and the second phase from the consumer, the actual construction of the views in both phases is done fully automatically. The two-phase approach supports the interests of both providers and consumers.

We now explain these two phases, aggregation and customization, in more detail, using the example in Figure 1. The internal, private process model is shown on the bottom while the customized process view is shown on the top. In the aggregation phase, the process provider first has to identify which activities of the business process have to be aggregated to hide private process details. In Figure 1, these activities to be aggregated are B and D, indicated with a double line. Next, the aggregate activity is constructed. This aggregate contains the identified activities, but possibly also some additional activities, to ensure that the derived process view is consistent with the underlying process model. In the non-customized process view in Figure 1, activity C has been added to the aggregate BCD. We define the procedure for constructing aggregate activities both declaratively (by construction rules) and operationally (by algorithms) and show the equivalence of both definitions. The declarative definition is useful to explain computed aggregates to end users, while the operational definition is more easy to implement. Finally, an external process view is constructed by replacing the aggregated activities with the aggregate. These steps can be repeatedly applied (not shown in the figure).

In the second phase, a consumer selects a set of activities that it wishes to see from the non-customized process view. In Figure 1 these activities are A and F, indicated with a double line. Next, fully automatically a customized process view containing these activities is constructed. All the other activities of the non-customized process view are either hidden or omitted in the customized view. In the customized view in Figure 1, activities BCD and E of the non-customized view are hidden by abstract activity BCDE.

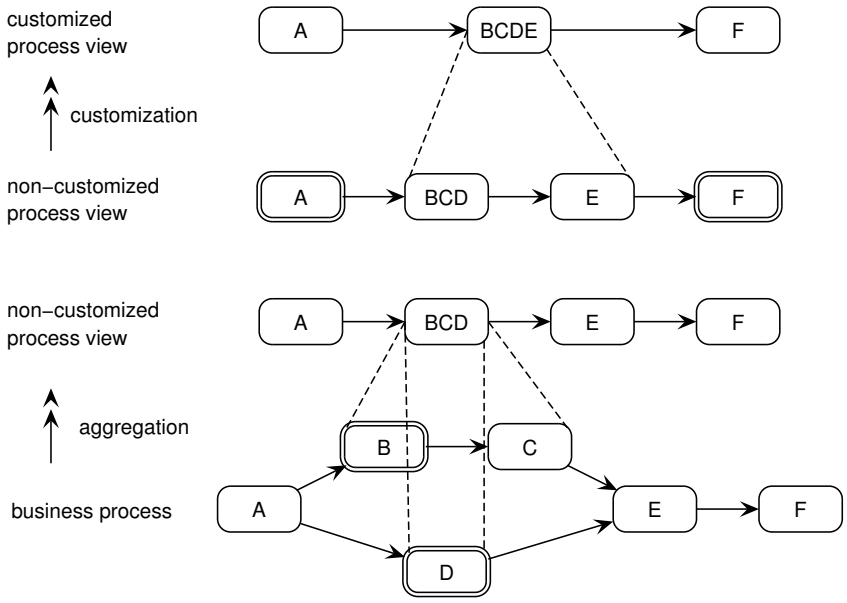

Fig. 1. Illustration of approach for generating customized process views

Customers can select the activities to be shown in the customized view by referring to organization-independent activities, which could come for example from industry standards like SCOR [27] or RosettaNet [25]. Providers can implement their own variant of such standard activities. To capture this, we use an inheritance relation on activities, to indicate that an activity at a company side inherits from (or implements) an abstract activity defined by an industry standard. For example, a consumer can request compound RosettaNet activity Request shipping order cancellation while the provider has implemented this activity as an activity Cancel order. The customized process view would then show Request shipping order cancellation instead of Cancel order. Using this inheritance-based approach, adherence to reference models is easily obtainable.

In the remainder of this paper, we focus on block-structured process models [9, 17, 20], or structured process models for short. In such models, each block has a unique entry and a unique exit point, and blocks are properly nested. If a structured process model is sequential, its structure is similar to that of a structured program. Many existing process description languages, including industry standard BPEL [3] and OWL-S [21], are structured into blocks. We show that the block-structure allows for a simple and efficient (tractable) procedure for constructing customized views. Unstructured process models may contain structural errors such as deadlocks, for example if an OR split is followed by an AND join. Block-structured process models require that splits and subsequent joins have the same type, and thus they do not have this drawback [17].

In sum, the main contribution of the paper is the definition of an approach that helps process providers and consumers to construct tailored process views on private processes in an efficient way taking process standards into account. Providers can construct process views by hiding those parts of their business processes that have to remain secret. Consumers can construct customized process views by hiding or omitting irrelevant parts of process views offered by the providers. This allows an efficient setup of dynamic business-to-business collaborations with a strong process-orientation [10]. Examples of such collaborations are virtual enterprises whose key operations are process-oriented, like logistics chains [11], financial and insurance networks [8], and industrial networks [7].

Structure. The remainder of this paper is organized as follows. Section 2 defines structured process models. Section 3 defines the first phase of the approach: how a non-customized view can be derived from a conceptual-level process, given a set of nodes to be aggregated. The construction procedure defines which nodes, next to the given nodes, need to be aggregated in order to get a process view that is consistent with the conceptual process. The construction procedure is defined both declaratively (by construction rules) and operationally (by algorithms) and the equivalence of both definitions is shown. Section 4 defines the second phase of the approach: how, given a set of abstract activities that a consumer wants to be visible, a process view can be customized to show only the relevant activities in full detail. The constructed customized view will contain concrete activities that are specializations of the abstract activities. Section 5 shows how our approach can be architecturally supported by extending an existing software architecture for setting up and enacting dynamic virtual enterprises. Section 6 discusses related work. Section 7 wraps up with conclusions and further work

2 Preliminaries

We first define structured process models and then fix an inheritance relation on activities. We also give some auxiliary definitions that we use in Section 3.

Structured Process Models. A process model specifies how a given set Act of activities (atomic unit of works) are ordered. The used ordering constructs are sequence, choice, parallelism, and structured loop. To simplify the exposition, we abstract from data.

Let $\mathcal{P}$ denote the set of all structured process models. A structured process model $P \in \mathcal{P}$ is a tuple $(A, N, \mathit{child}, \mathit{type}, \mathit{label})$ where

- $A \subseteq Act$ is a set of activities,
- N is a set of nodes,
- $\mathit{child} : N \times N$ is a predicate that defines the hierarchy relation. We have $\mathit{child}(n, n')$ if and only if n is a child (sub) node of n'.
- $\mathit{type} : N \longrightarrow \{SEQ, PAR, XOR, LOOP, BASIC\}$ is a function that assigns to each node its type. Type SEQ indicates that all children of the node execute in sequence, PAR that they execute in parallel, XOR that one of them is

executed at a time, and *LOOP* that the children execute zero or more times. We require that each *SEQ*, *XOR*, and *PAR* node has more than one child and that each *LOOP* node has only a single child, which is no *LOOP* node. A node has type *BASIC* if and only if it is a leaf node, i.e. it has no children.
- $label : N \longrightarrow A \cup \{\tau\}$ is a function labeling a node with an activity. Note that the same activity can label different nodes. The τ symbol is used to denote an aggregate activity, i.e., an activity in a process view that aggregates activities from a lower level[1].

We use an auxiliary function $children : N \longrightarrow \mathbb{P}\, N$ that defines for each node its set of child nodes. For a leaf node, this set is empty. The definition of *children* makes use of predicate *child*:

$$children(n) \stackrel{\text{df}}{=} \{n' \in N \mid child(n', n)\}.$$

If $c \in children(n)$, node n is parent of c, written $parent(c)$. By $children^+$ and $children^*$ we denote the irreflexive-transitive closure and reflexive-transitive closure of *children*, respectively. So $children^*(n) = children^+(n) \cup \{n\}$. If $n \in children^*(n')$, we say that n is a descendant of n' and that n' is an *ancestor* of n. Note that each node is ancestor and descendant of itself.

To ensure that the *child* predicate indeed arranges nodes in a hierarchy, we require that each node has one parent, except one node r, which has no parent. Next, we require that r is ancestor of every node in N. These constraints ensure that node are structured in a tree with root r. Leaves of the tree are the *BASIC* nodes. Internal nodes have type *SEQ*, *PAR*, *XOR*, or *LOOP*.

To indicate the ordering of children of nodes of type *SEQ*, we use a partial function $rank : N \longrightarrow \mathbb{N}$. The ranks of two nodes are only compared if the nodes share the same parent that has type *SEQ*. We require that two different nodes with the same parent have different ranks, and that for a node n with l children, for any child c of n, $rank(c) \in \{0, .., l-1)$. Using an overloading of notation, we use $rank(n, i)$, where $0 \leq i \leq l-1$, to indicate the unique child c of n for which $rank(c) = i$.

In the remainder of this paper, we will show structured process models graphically, using a variant of the UML activity diagram notation [28]. Figure 2 shows the business process of a logistics organization that delivers cellular phones (GSM phones) from a warehouse to a customer. We use node containment to indicate hierarchy. The root node is never shown. Sequence nodes have an incoming and outgoing arrow crossing their border, whereas choice and parallel nodes have a diamond and bar, respectively, on their border. Within a sequence node, the ordering relation is specified by means of arrows. Loop nodes have no dedicated symbol, but are indicated by drawing a self edge for the unique child of the loop node. For example, in Figure 2 node Pick GSM from stock is child of a loop node. To distinguish nodes from activities, in the remainder nodes are written

[1] The τ symbol comes from the field of process algebra, where it is used to denote an invisible action [22].

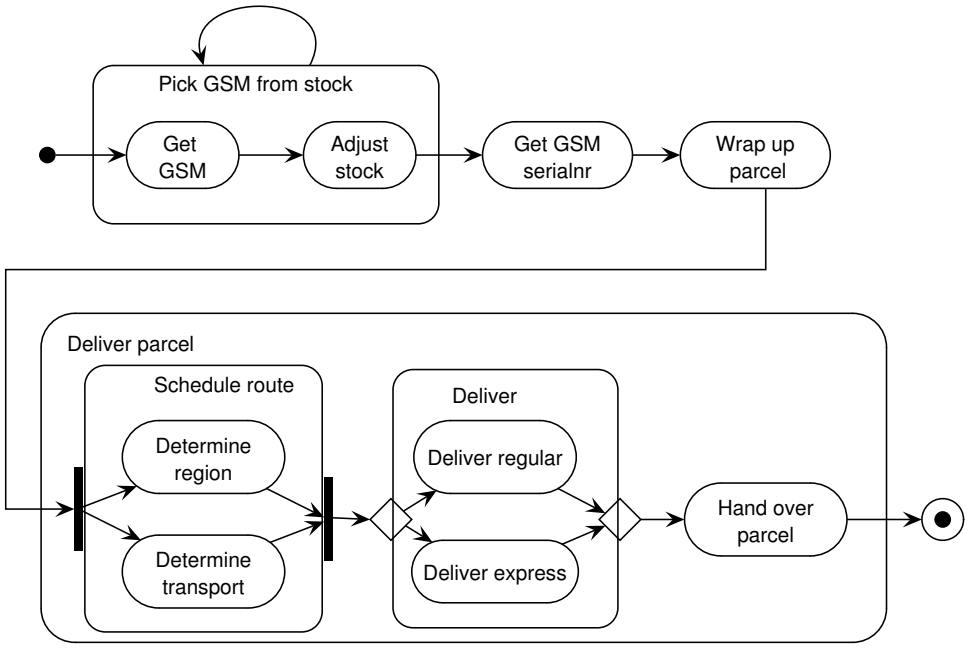

Fig. 2. Logistics process (adapted from [29])

in sans serif whereas non-τ activities are written in *italic*. To simplify the exposition, we assume in each example that the activity has the same name as the corresponding node.

Least common ancestors. To define the construction of process views in Section 3, we will make use of some auxiliary functions on the syntax of structured process models. The definitions are inspired by formal statechart semantics [15, 24].

For a set X of nodes, the least common ancestor (lca) of X, denoted $lca(X)$ is the node x such that x is ancestor of each node in X, and every other node y that is ancestor of each node in X, is ancestor of x:

- $X \subseteq children^*(x)$, and
- For every $y \in N$ such that $X \subseteq children^*(y)$, we have that $x \in children^*(y)$.

Since nodes are arranged in a tree, every set of nodes has a unique least common ancestor. For example, in Fig. 2 the lca of Deliver regular and Deliver express is Deliver, whereas the lca of Deliver regular and Hand over parcel is Deliver parcel. Note that the lca of a single node is the node itself, i.e. $lca(\{x\}) = x$.

Based on the notion of lca, we define some additional relations on nodes. The before relation $<$ denotes temporal ordering. Given two nodes $n, n' \in N$, we have n before n', written $n < n'$, if and only if

- node $l = lca(\{n, n'\})$ has type SEQ, and

– for the children $c_n, c_{n'}$ of l such that n is descendant of c_n and n' is descendant of $c_{n'}$, we have $rank(c_n) < rank(c_{n'})$.

For example, in Fig. 2 we have Determine region $<$ Deliver express.

Given two nodes $n, n' \in N$, we have n orthogonal to n', written $n \perp n'$, if and only if $n \neq n'$ and the type of $lca(\{n, n'\})$ is not SEQ. Since we require that a node of type $LOOP$ has only a single child, if $n \perp n'$ then their lca is either a PAR or a XOR node. For example, in Fig. 2 we have Deliver regular$\perp$Deliver express.

Inheritance of activities. Above we introduced a set Act of activities. Let $\leq \subseteq Act \times Act$ be an inheritance relation on activities. If $a \leq a'$ than a is more specific than a' and a can replace a', since a has all features of a'. For example, activity *Handover parcel* from Figure 2 inherits from SCOR [27] activity *Receive and Verify Product at Customer Site*. Relation $\leq$ is a partial order, so if $a \leq b$ and $b \leq a$ then $a = b$. We allow multiple inheritance, so it might be that $a \leq b$ and $a \leq c$ yet b and c are incomparable, so $b \not\leq c$ and $c \not\leq b$. For example, in SCOR [27] we have that activity *Enable return* is a specialization of both *Enable* and *Return*, but these activities are incomparable. Since $\leq$ is a partial order, the inheritance relation is acyclic, so an activity cannot inherit indirectly from itself.

3 Constructing Process Views

This section defines the first phase of our approach: how process views can be constructed from structured process models. First, we define how a given set of nodes from the process model can be aggregated in a correct way into a single node in the process view. Second, we define how, given a computed aggregate and a structured process model, a structured process view can be derived. These two steps can be repeated arbitrarily often, so a process view can itself be further aggregated into a more abstract process view.

3.1 Constructing aggregates

An aggregate is a set of nodes from the process model that is represented in the process view by a single node n , i.e. node n hides the nodes contained in the aggregate. The user must specify which set of nodes has to be aggregated. However, the aggregate might need to contain some additional nodes as well, in order to get a process view that is consistent with the underlying process model. The view and the process model are consistent if the orderings of the process model are respected by the view and no additional orderings are introduced in the view. We specify the consistency constraints as construction rules, to be satisfied by the constructed aggregated. Next, we define an algorithm for constructing aggregates. We show that both definitions are equivalent.

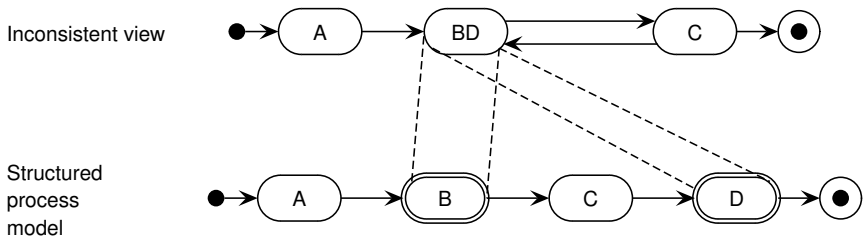

Fig. 3. Example to motivate Rule 2

Construction rules. Let X be the set of conceptual-level nodes that have to be aggregated. Denote by $agg(X)$ the set of nodes that the aggregate constructed for X should contain in order to derive a process view consistent with the underlying process model. Naturally, all nodes of X should be in $agg(X)$, which leads to the first rule for constructing aggregates:

$$\text{Rule 1} \qquad X \subseteq agg(X)$$

The other rules are defined to ensure that after constructing the aggregate, the resulting process view is consistent with the underlying structured process model. Rule 2 states that if two nodes x, y are aggregated such x is before y, then every intermediary node i, so $x < i < y$, should be contained in the aggregate as well. Otherwise, if an intermediary node i is not included, the aggregate will not be atomic anymore in the process view. Then the aggregate will be on the one hand before $agg(X)$, since $x < i$, but also after $agg(X)$, since $i < y$. For example, aggregating in Figure 3 nodes B and D without aggregating C would result in a view in which the aggregate is before and after C. Thus, a loop is created which is not present in the original model. Therefore, we require that i is included in the aggregate:

$$\text{Rule 2} \qquad \text{if } x, y \in agg(X) \text{ and } i \in N \text{ such that } x < i < y \text{ then } i \in agg(X).$$

Rule 3 states that if a composite node is included in the aggregate, all its children are included as well. This ensures that in the process view aggregates have no children, i.e. no internal details of the aggregate are revealed. For example, if in Figure 4 node X is to be aggregated, then B and C must be aggregated as well. Otherwise, the process view would be the same as the original process, with X replaced by τ.

$$\text{Rule 3} \qquad \text{if } x \in agg(X) \text{ then } \mathit{children}(x) \subseteq agg(X).$$

Rule 4 requires that if a node is in the aggregate and its parent is a strict descendant of $lca(X)$, so $x \neq lca(X)$, then its parent node has to be aggregated

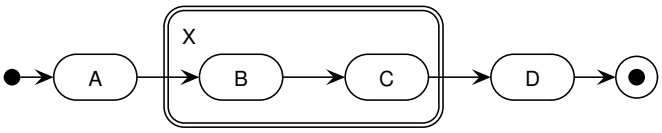

Fig. 4. Example to motivate Rule 3

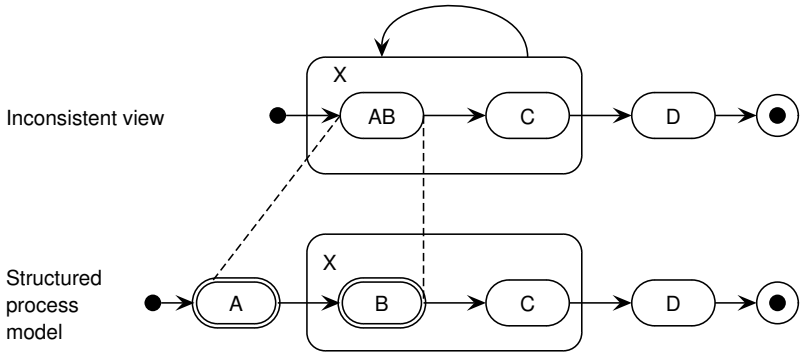

Fig. 5. Example to motivate Rule 4

as well. Not using this rule can lead to inconsistent process views. For example, Figure 5 shows a view constructed by aggregating A and B without aggregating X, even though X is strict descendant of $lca(\{\mathsf{A},\mathsf{B}\})$, which is root r. Since in the structure process model node A is before X, in the view AB is before X. Since X contains AB, then there is a self loop for X. This loop is not present in the original process model, so the view is not consistent.

To see why the parent node needs to be a strict descendant of $lca(X)$, rather than a descendant, consider the example in Figure 6. If A and D are to be aggregated, then $lca(\{\mathsf{A},\mathsf{D}\}) = \mathsf{Z}$ should not be aggregated. Otherwise, in that case the parallel branch Y would be aggregated as well. As the view shows, however, that branch can still be exposed.

$$\text{Rule 4} \qquad \begin{array}{l}\text{if } x \in agg(X) \text{ and } parent(x) \in children^{+}(lca(X)) \\ \text{then } parent(x) \in agg(X)\end{array}$$

Note that for a given set X of nodes to be aggregated in a structured process model P, there can be multiple sets which satisfy the construction rules. We formally define the notion of a minimal aggregate, which is the minimal set satisfying the four construction rules. Given a set X of nodes to be aggregated in process model P, the minimal aggregate for X, written $min\_agg(P, X)$, is the smallest set S satisfying:

– $X \subseteq S$

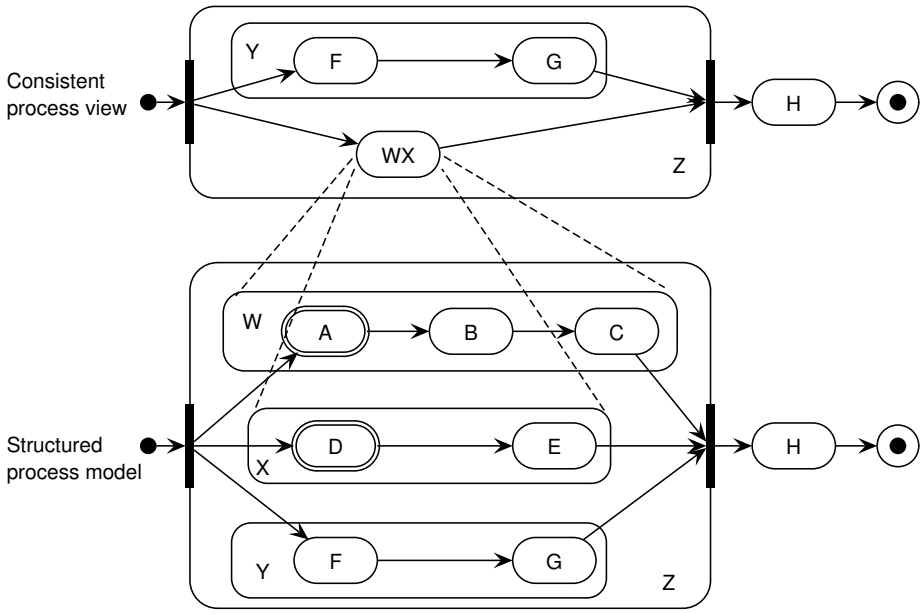

Fig. 6. Another example to motivate Rule 4

- if $x, y \in S$ and $i \in N$ such that $x < i < y$ then $i \in S$
- if $x \in S$ then $\mathit{children}(x) \subseteq S$
- if $x \in S$ and $\mathit{parent}(x) \in \mathit{children}^+(\mathit{lca}(X))$ then $\mathit{parent}(x) \in S$

This declarative definition suggests an iterative algorithm for computing the minimal aggregate for a set of nodes X: start with the nodes in X, and repeatedly add nodes to resolve violations of the last three construction rules. However, this requires computation of the $<$ and $\perp$ relations, which is cumbersome. We therefore define a more efficient algorithm for computing minimal aggregates.

Algorithm. The algorithm AGGREGATE for constructing an aggregate is listed in Fig. 7. It expects a structured process model P and a set X of nodes to be aggregated, and returns an aggregate.

The algorithm first computes the least common ancestor l of the set X of nodes to be aggregated. If $l \in X$ then all descendants of l need to be aggregated (l. 4). Otherwise, all children of l which have some descendant in X are put in set C (l. 6). Next, the set of all descendant of nodes in C are put in aggC (l. 7). These descendants are to be contained in the aggregate by Rule 3 and Rule 4.

Next, the type of l is tested (l. 8):

- If l has type SEQ, then the set C' of "intermediate" children of n are computed. Set C' contains a child c of n if and only if there are children $c_1, c_2 \in C$ such that $c_1 < c < c_2$. The set of all descendant of nodes in C'

```
 1: procedure AGGREGATE(P, X)
 2:     l := lca(X)
 3:     if l ∈ X then
 4:         agg := children*(l)
 5:     else
 6:         C := {c ∈ children(l) | X ∩ children*(c) ≠ ∅}
 7:         aggC := ⋃_{c∈C} children*(c)
 8:         if type(l) = SEQ then
 9:             C' := {c ∈ children(l) | ∃ a, b ∈ C : a < c < b}
10:             aggC' := ⋃_{c'∈C'} children*(c')
11:             agg := aggC ∪ aggC'
12:         else
13:             agg := aggC
14:         end if
15:     end if
16:     return agg
17: end procedure
```

Fig. 7. Algorithm for constructing aggregates

is put in $aggC'$ (l. 10). All descendants of these intermediate children of n must be contained in the aggregate by Rule 2 and Rule 3. The aggregate is therefore the union of sets $aggC$ and $aggC'$ (l. 11).

– If l has type PAR or XOR, then the aggregate only contains all descendants of C (l. 13).

Finally, the constructed aggregate is returned (l. 16).

Note that computing the least common ancestor can be done in linear time [16]. Using this observation, it is easy to see that the algorithm is tractable (polynomial).

Correctness. The correctness of the algorithm is shown by the following theorem, which states that the algorithm yields a minimal aggregate that satisfies all four construction rules.

Theorem 1. *Given a conceptual process model P and a set X of nodes to be aggregated. Then $min\_agg(P, X) = \text{AGGREGATE}(P, X)$.*

Proof. Denote by agg the set returned by $\text{AGGREGATE}(P, X)$. We prove the claim in two steps: first we show that agg complies with the construction rules (i), and next we show that agg is minimal (ii).

i. We only show the proof for Rule 2; the proofs for the other rules are by similar reasoning.

Let $x, y \in agg$ such that there is a $z \in N$ with $x < z < y$. We will show that $z \in agg$.

Since $x < z < y$, we have $l_{x,y,z} = lca(\{x, y, z\})$ is of type SEQ. If $l_{x,y,z} = l$, by l. 11 of AGGREGATE, we have $z \in agg$. If $l_{x,y,z}$ is a strict descendant of l, so $l_{x,y,z} \neq l$, there are two cases:

- l is of type *PAR* or *XOR*. Let c be the child of l that is ancestor of $l_{x,y,z}$. Then $c \in C$ and so $z \in agg$ by l. 13.
- l is of type *SEQ*. Let c_z be the child of l such that z is descendant of c_z. Since $x < z < y$, $c_z \in C \cup C'$. The claim then follows from l. 11.

ii. We prove the claim by contradiction. Suppose there is another set agg', with $agg' \subset agg$, such that $X \subseteq agg'$ and agg' does not violate the construction rules 2, 3, and 4.

Let n be a node in $agg \setminus agg'$, so n is superfluously added to agg by the algorithm. We now examine the cases where n might have been added to agg by the algorithm:

- l. 4. Since $l \in X$, by assumption $l \in agg'$. Let n' be the node in $agg \setminus agg'$ such that n' is descendant of l and n' is ancestor of n and $parent(n') \in agg'$. Since $n' \notin agg'$, Rule 3 is violated for agg'.
- l. 11. Then the lca of X is of type *SEQ*. Then n is a descendant of either some child $c' \in C'$ (l. 10) or of some child $c \in C$ (l. 7). In the first case, $n \notin agg'$ violates Rule 2. In the second case, there is a node $x \in X$ such that x is descendant of c (l. 6). Since $n \in agg \setminus agg'$, $n \neq x$. There are three subcases now
  - n is strict descendant of x. Let n' be the node in $agg \setminus agg'$ such that n' is a strict descendant of x and ancestor of n and $parent(n') \in agg'$. Since $n' \notin agg'$, Rule 3 is violated.
  - n is strict ancestor of x. Let n' be the node in $agg \setminus agg'$ such that n' is descendant of n and a strict ancestor of x and $children(n') \cap agg' \neq \emptyset$. Since $n' \notin agg'$, Rule 4 is violated.
  - $n < x$ or $n < x$ or $n \perp x$. Let $l_{n,x} = lca(\{n, x\})$. Since $l_{n,x}$ is a strict ancestor of x but a descendant of c, node $l_{n,x}$ is in agg' by the previous case. Let n' be the node in $agg \setminus agg'$ such that n' is a (strict) descendant of $l_{n,x}$ and $parent(n') \in agg'$. Since $n' \notin agg'$, Rule 3 is violated.
- l. 13. Let $x \in X$ be a node such that $x \in aggC$. By assumption (Rule 1), $x \in agg'$. So $n \neq x$. Let c_x be the child of l that is ancestor of x.
  - n is strict descendant of x. Let $n' \in agg \setminus agg'$ be a descendant of x that is ancestor of n such that $parent(n') \in agg'$. Since $n' \notin agg'$, then agg' violates Rule 3.
  - n is strict ancestor of x. Let $n' \in agg \setminus agg'$ be a descendant of n that is ancestor of x such that $children(n') \cap agg' \neq \emptyset$. Since $n' \notin agg'$, then agg' violates Rule 4.
  - $n \perp x$. Let $l_{n,x} = lca(\{n, x\})$. Since $l_{n,x}$ is strict ancestor of x, by the previous case $l_{n,x} \in agg'$. Let $n' \in agg \setminus agg'$ be a descendant of $l_{n,x}$ that is ancestor of n such that $parent(n') \in agg'$. Since $n' \notin agg'$, then agg' violates Rule 3.

Thus, in each case agg' violates a construction rule, which contradicts the assumption. □

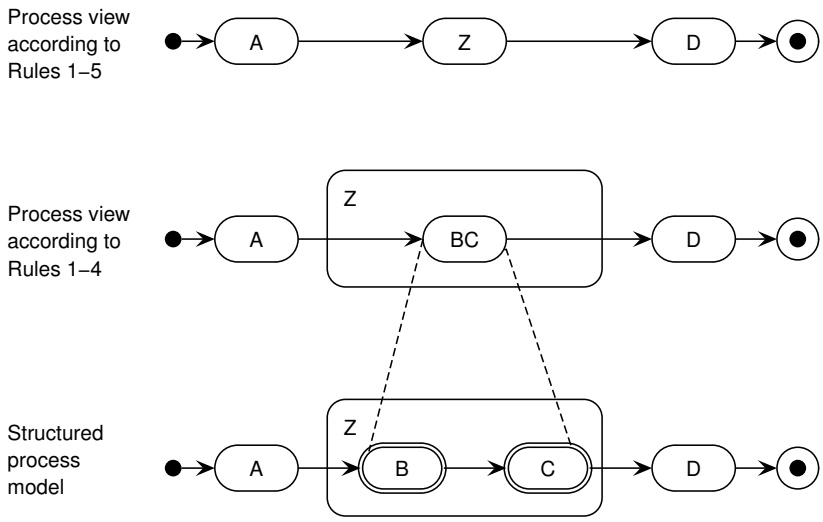

Fig. 8. Example to motivate Rule 5

3.2 Extension

Given a set X of nodes to be aggregated, the least common ancestor $lca(X)$ is only aggregated if $lca(X) \in X$. Sometimes, this can lead to process views which are correct, but counter intuitive. For example, aggregating B and C in Fig. 8 according to the four construction rules results in the middle process view. Here, Z is not aggregated, because $\mathsf{Z} = lca(\{\mathsf{B}, \mathsf{C}\})$. All of the children of Z are aggregated, so Z only has a single child, which is the new aggregate. But in this case it would make sense to aggregate Z, since all of the children of Z are aggregated as well.

Therefore we define an additional rule, which states that if all children of $lca(X)$ are included in the aggregate, node $lca(X)$ itself should be included as well.

$$\text{Rule 5} \qquad \text{if } \mathit{children}(lca(X)) \subseteq \mathit{agg}(X) \text{ then } lca(X) \in \mathit{agg}(X)$$

The algorithm needs to be modified slightly to take into account this new construction rule. Between lines 14 and 15, the following lines needs to be inserted:

if $\mathit{children}(l) \subseteq \mathit{agg}$ **then**
  $\mathit{agg} := \mathit{agg} \cup \{l\}$
end if

The theorem of the previous subsection can be easily extended to deal with this new construction rule and the modified algorithm.

3.3 Generating process views

Above, we have outlined a declarative and operational approach for constructing an aggregate, which is a set of nodes that have be represented by a single node in the process view. Now we define a function $gen : (\mathcal{P} \times N) \longrightarrow \mathcal{P}$ that generates from a given structured process model and an aggregate the resulting process view, which is again a structured process model. If there are multiple aggregates, the function can be repeatedly applied.

If agg is the constructed aggregate for process model $P = (A, N, child, type, label)$, so $agg \subseteq N$, then the process model $P' = gen(P, agg)$ is constructed by replacing agg with a new node $n_{agg} \notin N$ that does not get any children in the process view P' and gets label τ.

Now the problem is that the new node n_{agg} needs to be attached as child to some node $N \setminus agg$, i.e., some node $l \in N \setminus agg$ has to act as parent of n_{agg} in the process view P'. Let l be the lowest node in $N \setminus agg$ that is ancestor (in P) of all nodes in agg. So $agg \subseteq children^*(l)$ and for every other node $l' \in N \setminus agg$ for which $agg \subseteq children^*(l')$, we have $l \in children^*(l')$. From the algorithm, it follows that if $lca(agg) \notin agg$ then $l = lca(agg)$; otherwise, l is the parent of $lca(agg)$ (l. 4). Thus, the construction procedure ensures that node l exists and is unique. Therefore, l can be the unique parent of n_{agg} in P'.

Formally, $P' = (A', N', child', type', label')$ where

- $A' = \{y \mid (x, y) \in label'\}$
- $N' = N \setminus agg \cup \{n_{agg}\}$
- $child' = (child \cap (N' \times N')) \cup \{(n_{agg}, l)\}$
- $type' = type \cap (N' \times \{SEQ, PAR, XOR, BASIC\}) \cup \{(n_{agg}, BASIC)\}$
- $label' = (label \cap (N' \times A)) \cup \{(n_{agg}, \tau)\}$

Instead of labelling the new aggregate node with τ, the user (provider) can also decide to create a new activity. In the examples shown in this section, each aggregate of a consistent process view was labelled with a new activity, rather than τ.

Figure 9 shows the process view for Figure 2 if Determine region and Deliver express are selected for aggregation.

4 Customizing Process Views

This section defines the second phase of our approach: how constructed process views can be customized. Input is a process view $P \in \mathcal{P}$ plus a set I of activities which the consumer wishes to be visible for monitoring the progress of the process. We require that all activities in I are incomparable, so there are no two activities such that one is descendant of the other. Output is a customized view, which is again a structured process model $P' \in \mathcal{P}$. In P' irrelevant parts of P with respect to I are omitted or aggregated. A part is irrelevant if none of the contained nodes executes an activity that implements some activity in I. Not every activity $i \in I$ needs to be implemented in P'. However, the approach can be easily modified and extended to deal with additional constraints.

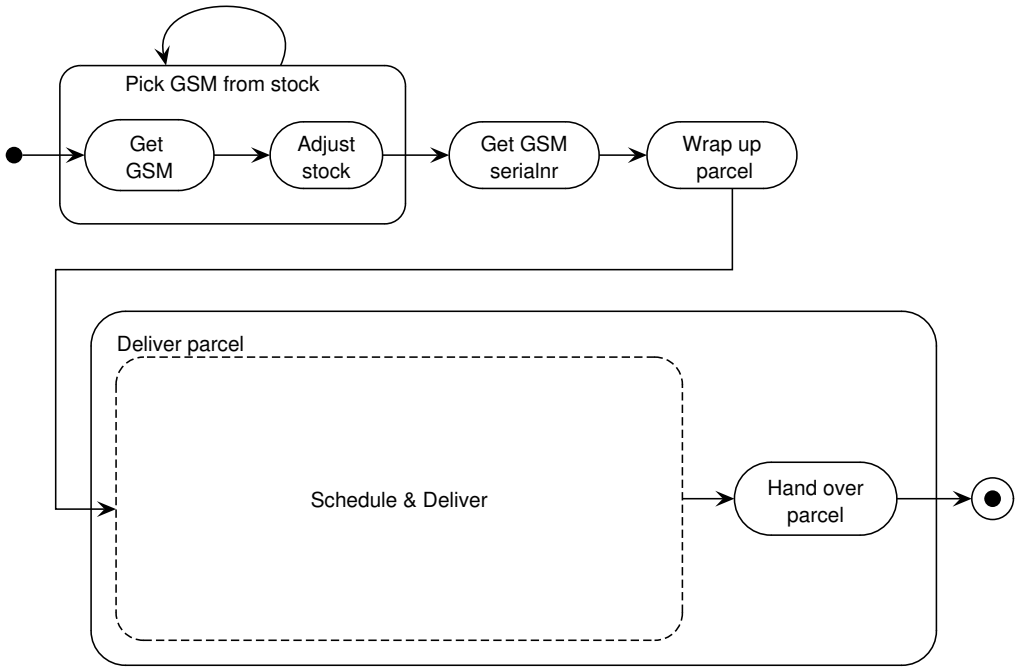

Fig. 9. Example process view for Figure 2

We define the customization algorithm declaratively as a function, *customize* : $(\mathcal{P} \times \mathbb{P}\, Act) \longrightarrow \mathcal{P}$, which transforms a structured process model P and a given set I of activities into a structured process model $customize(P, I) = P'$. We now define the individual components of $P' = (A', N', child', type', label')$, given the input model $P = (A, N, child, type, label)$.

The set A' of activities contains the requested activities in I plus the activities from A that are actually used in the new labelling function $label'$, which is defined later:

$$A' = \{\, a \mid a \in A \cup I \wedge \exists\, n \in N' : (n, a) \in label' \,\}.$$

Before we define N', we fix some terminology. A node n is *relevant* if one of its descendant nodes is labelled with an activity that implements an activity $i \in I$:

$$relevant(n, I) \overset{\text{df}}{\Leftrightarrow} \text{there is a } n' \in children^*(n) \text{ and } i \in I : label(n') \leq i.$$

For example, for Figure 2, if $I = \{\mathsf{Determine\ Transport}\}$ then among others node Deliver parcel is relevant.

When customizing a process view, we have to ensure that each relevant node in the process view occurs in the customized view, since the external party wishes

to monitor relevant nodes. So all relevant nodes of P should be in P'. The set of relevant nodes in the customized view is defined as:

$$N'_{rel} = \{n \in N \mid \mathit{relevant}(n, I)\}.$$

Naturally, N' should contain N'_{rel}. However, N' should also contain new abstract nodes, not in N, that hide irrelevant nodes in N. These abstract nodes are needed in P' to get a valid process view. For example, in the customized view in Fig. 1 there is an abstract node BCDE that hides irrelevant nodes B, C, D, and E. Omitting BCDE would result in an invalid process view.

An abstract node only has to be created to hide the irrelevant children of a compound node that also has relevant children. For example, in Figure 1 the abstract node BCDE is created for compound node root r, which has both relevant and irrelevant children. If a relevant node has only irrelevant children, it needs no children in the customized view. If a relevant node has only relevant children, all these children are shown in the customized view. So in these last two cases, an abstract child node is not needed.

The predicate *relevant_compound* formally defines for which compound nodes of N an abstract child node needs to be created:

$$\begin{aligned}\mathit{relevant\_compound}(n, I) \Leftrightarrow\ & \mathit{relevant}(n, I) \\ & \wedge\ \exists\, n' \in \mathit{children}(n) : \mathit{relevant}(n', I) \\ & \wedge\ \exists\, n' \in \mathit{children}(n) : \neg\ \mathit{relevant}(n', I).\end{aligned}$$

Note that $\mathit{relevant\_compound}(n, I)$ is true implies n is compound, since only compound nodes have children.

We now define per type t of compound nodes, which concrete nodes of this particular type get a new abstract child in the customized view to hide irrelevant children from the input model. These newly created nodes are put in set N'_t. More precisely, if $n \in N$ is a node such that $\mathit{relevant\_compound}(n, I)$, then we denote by $n_t \in N'_t$ a new (fresh) abstract node, so $n_t \notin N$. In the customized view, n_t hides all irrelevant descendants of n. However, not every relevant compound node n needs such a node n_t. For certain types of nodes, irrelevant child nodes of a relevant compound node can also be completely omitted, rather than being represented by some abstract node.

Before we define N'_{XOR}, we observe that omitting irrelevant children from a relevant *XOR* node can result in a model with illegal states. For example, suppose for Figure 2 the external party wishes to monitor node Deliver express. Omitting in the customized process view Deliver regular would result in an illegal state at the external level if in the internal state Deliver regular were executed. The external state would show that *XOR* node Deliver is performed, but the external party can also see that none of its children (i.e., Deliver express) is performed. So for the irrelevant children, i.e. Deliver regular, a new abstract node must be created in the customized process view, say Other delivery, i.e., non-express delivery. Therefore, for each relevant *XOR* node n a new abstract node n_{xor} is created, which leads to the following definition of N'_{XOR}:

$$N'_{XOR} = \{\, n_{xor} \mid n \in N \wedge \mathit{type}(n) = \mathit{XOR} \wedge \mathit{relevant\_compound}(n, I)\,\}.$$

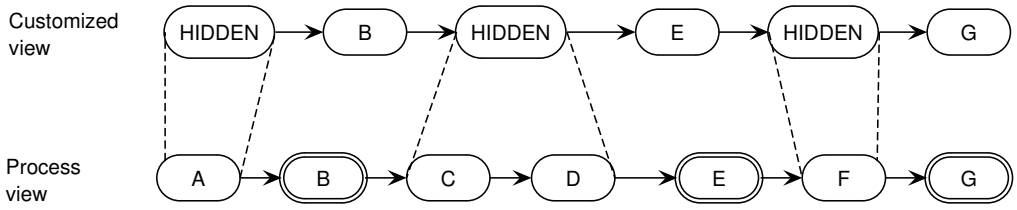

Fig. 10. Example to illustrate the customization of sequential nodes

The parents of the newly created nodes in N'_{XOR} are defined below by the *child'* relation.

For a relevant *PAR* node, in principle both abstraction and omission of irrelevant children are possible. Omission is possible because the execution states are still well defined through the children that are relevant. In this paper, we choose to omit these irrelevant children, to show as little detail as possible, but the alternative (abstraction) can be easily defined. Thus, each relevant parallel node has no (abstracted) irrelevant children in the customized view. For example, if in Figure 2 the consumer wishes to monitor activity *Determine transport*, node Determine region can be safely omitted, because the execution state is well defined through Determine transport. Consequently, each relevant *PAR* node does not need an abstract child node in the customized view, and therefore set N'_{PAR} is empty:

$$N'_{PAR} = \varnothing.$$

For sequential nodes, the situation is more complex. Given a relevant sequential node, omission of its irrelevant children is not possible, since then not all internal states would have valid external representations. Only abstraction is therefore feasible. For a sequential node that has some relevant children, all irrelevant children cannot be grouped into one node, since there might be an intermediate relevant child. Thus, only irrelevant children which are not interrupted by relevant children can be grouped (see Figure 10). For each *SEQ* node n, we therefore create for each maximal interval (i, j) of irrelevant children an abstract node $n_{seq(i,j)}$. In an interval (i, j) of irrelevant child nodes, each node $rank(n, k)$ is irrelevant, for $i \leq k \leq j$. In a maximal irrelevant interval, either $rank(n, i)$ is the first node of n, so $i = 0$, or node $rank(n, i-1)$ is relevant, and either $rank(n, j)$ is the last node of n, so $j = |children(n)| - 1$, or $rank(n, j+1)$ is relevant.

The set N'_{SEQ} of abstract child nodes created to hide irrelevant children of *SEQ* nodes in the customized view is therefore defined as:

$$\begin{aligned} N'_{SEQ} = \{\, n_{seq(i,j)} \mid\ & n \in N \wedge type(n) = SEQ \wedge relevant\_compound(n, I) \\ & \wedge\ 0 \leq i < j \leq |children(n)| - 1 \\ & \wedge\ (i = 0 \vee relevant(rank(n, i-1), I)) \end{aligned}$$

$$\wedge\ (j = |children(n)| - 1 \vee relevant(rank(n, j+1), I))$$
$$\wedge\ \forall\, i \leq k \leq j : \neg\ relevant(rank(n, k), I)\ \}.$$

Note that irrelevant begin and start activities of a sequence node are aggregated. If these irrelevant parts are at the top-level of the process, they can be omitted.

Finally, we have to define N'_{LOOP}. Since a *LOOP* node n has a single child, we have $\neg\ relevant\_compound(n, I)$. Thus, a *LOOP* node has does not have an aggregate child in the customized view, and therefore set N'_{LOOP} is empty:

$$N'_{LOOP} = \emptyset.$$

We now define N' as the union of N'_{rel} plus the sets containing all new abstract nodes:

$$N' = N'_{rel} \cup N'_{XOR} \cup N'_{PAR} \cup N'_{SEQ} \cup N'_{LOOP}.$$

Next, we have to define the remaining three predicates and functions. Predicate $child'$ is defined by restricting *child* to relevant nodes in N'_{rel} and adding for each *XOR* and *SEQ* node the constructed aggregate child nodes:

$$\begin{aligned} child' = &\{\ (c, p) \mid (c, p) \in child \wedge c \in N'_{rel} \wedge p \in N'_{rel}\ \} \\ &\cup \{\ (n_{xor}, n)\ \mid\ n_{xor} \in N'_{xor} \wedge n \in N'_{rel}\ \} \\ &\cup \{\ (n_{seq(i,j)}, n)\ \mid\ n_{seq(i,j)} \in N'_{seq} \wedge n \in N'_{rel}\ \}. \end{aligned}$$

Function $type'$ is defined to be the same as *type* for relevant nodes. Constructed abstract nodes have no children, so they are *BASIC*.

$$\begin{aligned} type' = &\{\ (n, t) \mid n \in N'_{rel} \wedge t = type(n)\ \} \\ &\cup \{\ (n, BASIC) \mid n \in N'_{XOR} \cup N'_{SEQ}\ \}. \end{aligned}$$

Finally, function $label'$ labels each relevant node $n \in N'_{rel}$ either with an activity in I or in A. A relevant node n is labelled with an activity $i \in I$ if and only if $label(n)$ implements i. This way, the consumer will see the activities on the level of detail it requires. If $label(n)$ does not implement i, node n gets labelled with $label(n)$. The constructed abstract nodes have no visible activity, i.e. they are labelled τ. Alternatively, the user (consumer) can create a new activity for a new abstract node.

$$\begin{aligned} label' = &\{\ (n, i) \mid n \in N'_{rel} \wedge i \in I \wedge label(n) \leq i\ \} \\ &\cup \{\ (n, l) \mid n \in N'_{rel} \wedge l = label(n) \wedge \nexists i \in I : label(n) \leq i\ \} \\ &\cup \{\ (n, \tau) \mid n \in N'_{agg}\ \}. \end{aligned}$$

Finally, we illustrate the customization algorithm by means of an example. If the process view in Figure 9 is customized for activities *Get GSM* and *Hand over parcel*, the customized view shown in Figure 11 is obtained. The label HIDDEN stands for τ. Note that Get GSM serialnr and Wrap up parcel have been merged into one abstract node in the customized view.

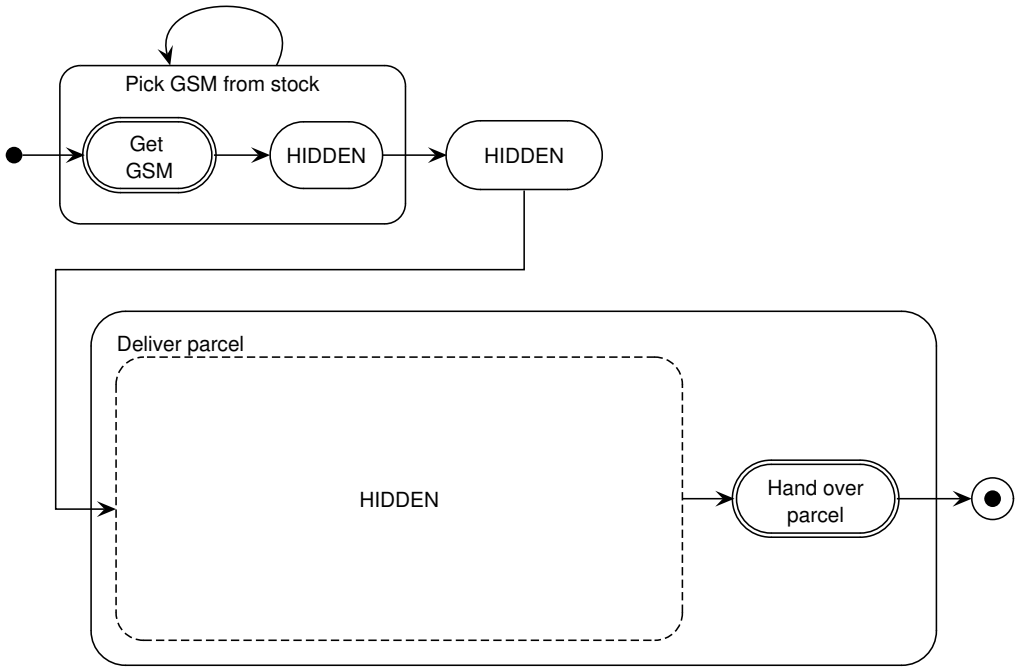

Fig. 11. Example customized process view for Figure 9

5 An Architecture for Constructing Process Views

In this section we explain how the approach can be supported by an architecture. The architecture supports the construction of views, not the actual enactment. This latter aspect is addressed in other work [26]. The proposed architecture is an extension of an existing architecture that supports the formation and enactment of dynamic virtual enterprises [14].

Before we show the architecture, we give a straightforward functional architecture that supports the construction of customized views from structured process models (Fig. 12). The aggregator and customizer module support the first and second phase, respectively, of the construction approach. The repository with the structured process models and the aggregator are located at the provider's site. The repository with the non-customized process views is a public

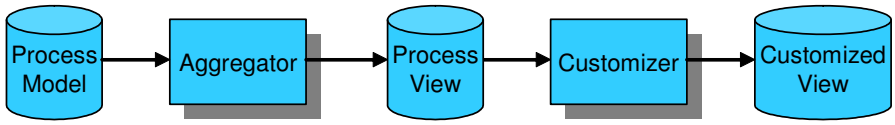

Fig. 12. Functional architecture for generating customized process views

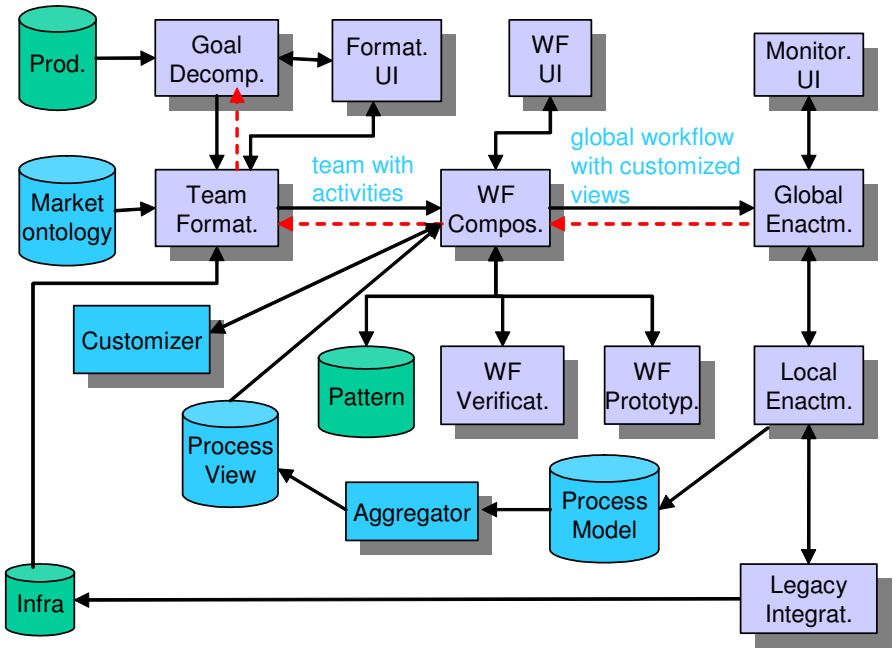

Fig. 13. Constructing customized views in CrossWork architecture

repository accessible to consumers. It can be located either at a provider site or at some public site. The customizer module and the repository containing the customized views can be located at the provider's site, but they can also be public components.

We now show how this generic functional architecture can be embedded in an existing architecture for forming and enacting virtual enterprises using agents and cross-organizational workflows. This existing architecture was created in the scope of the IST CrossWork project [7], which developed IT support for the dynamic formation of Networks of Automotive Excellence. Each network consists of a number of suppliers of moderate size. Together, the suppliers form a network (virtual enterprise) that can deliver to an Original Equipment Manufacturer (OEM) like BMW or MAN.

Figure 13 shows the extended CrossWork architecture. The basic CrossWork system has the following functionality. The system starts with the goal decomposition module. This module takes an order specification from an OEM and decomposes it into a required set of components and services, using a product knowledge base. Next, the team formation module finds for each identified component and service a partner using a market and infrastructure knowledge base [4]. The market knowledge base stores per organization the services and components it delivers and which activities it offers, while the infrastructure knowledge base stores information about the legacy systems of organizations.

The team formation module composes the retrieved partners into a team that can cooperate according to the market and infrastructure knowledge base. Then the workflow formation module queries the team members for their local workflow models (not shown) and composes these into a global process model. The ordering constructs used to compose the workflows are based on workflow patterns [2]. This constructed global workflow model can be verified and validated. Finally, the global workflow model can be enacted. The global workflow coordinates the local workflows of the team partners. The process language used to model processes is a structured process language based on XRL [1]. For the enactment, this language is mapped to BPEL [3].

The following extension of the CrossWork architecture supports the construction of customized views. The market knowledge base stores for each organization which organization-independent (e.g. SCOR [27], RosettaNet [25]) activities it offers. Each potential team partner stores the description of its local workflows in a private repository and can construct process views using the aggregator module. The constructed views can be stored locally at each provider's site or in a public repository. The workflow formation module can retrieve these constructed views as input by querying the team partners or searching the repository. Before the composition is started, however, the process views can be customized using a list of organization-independent activities that the team is supposed to implement. This list is provided by the team formation module. Then, the workflow formation module can compose the customized views into a global workflow. To support the enactment of the local views, also extensions are needed but these are outside the scope of this paper. See for example [26] for an existing approach that supports the enactment of process views.

6 Related Work

The most relevant work related to ours is that by Liu and Shen [18, 19]. In [18], they focus on deriving a process view from a given structured process definition. They concentrate too on defining for a conceptual process definition aggregate activities (virtual in their terminology) from a given set of conceptual-level activities (called essential). However, these activities must be basic, so they cannot contain other activities. They define rules on aggregates to ensure consistency between an external process view and a conceptual process model. They also present an algorithm that derives a minimal virtual activity from a set of essential activities. Based on this algorithm, from a given structured process model, a set of virtual activities can be derived that appear in the process view. The ordering of these virtual activities in the view can be inferred from the underlying process model. Next, they show that some consistency requirements between the process view and the original process hold. This work is extended in [19] by considering data flow as well.

This paper improves and extends the work of Liu and Shen [18] in several ways. Basically, their work coincides with our first phase. However, our formalization of structured process models is much more simple than theirs, leading to

more simple construction rules and a more efficient algorithm as well. Especially their consistency rules for models containing loops are quite complex, and their algorithm has a high complexity (it contains two nested loops). Next, we support the aggregation of composite nodes, while they only focus on aggregation of activities which have no subactivities. In addition to improving their work, we extend their work by allowing views to be customized. Customization involves inheritance of activities and omission of activities, something which is not considered by Liu and Shen. To simplify the exposition, we have ignored data flow, but it can be incorporated along similar lines as described in [19].

Chiu et al. [6] use process views to support interoperability of multiple workflows across organizations. They present a meta model and an interoperation model for workflow views, consisting of communication scenarios between these views, and a set of interoperation parameters. Consistency constraints are described to ensure that a workflow view is consistent with its underlying workflow, and that the communication between workflow views, as specified by communication scenarios, is consistent. Finally, they show how the approach can be realized using web services and XML technology. There are several differences with our work. First, we focus on the actual construction of consistent customized process views from a given business process, while they focus on consistency of a given workflow view and a given workflow. Second, they do not consider customization of views. Third, they consider unstructured process models while we focus on structured ones.

Next, there are approaches that use views for enabling inter-organizational workflow cooperation [5, 30]. The approach of Chebbi et al. [5] consists of three main steps: workflow advertisement, workflow interconnection, and workflow cooperation. The main focus of the paper is on the second step. They present reduction rules to derive from an internal process model an abstract process model which only contains tasks that cooperate with partner workflows outside the organization. On this public process, partner-specific views can be defined that are linked with an access contract. Zhao et al. [30] use visibility constraints on internal process models to derive partner-specific workflow views. Each partner can combine the workflow views of its partner with its internal process into what Zhao et al. call a relative workflow model. Next, they discuss how an organizational can use a relative workflow model to track the progress of its indirect partners, e.g. how a consumer can track the progress of the process of the provider of the provider.

There are several differences between [5, 30] and our work. First, they do not consider consistency criteria between a public process and internal one, whereas we have formalized these criteria as construction rules. Consequently, they do not give any proof of correctness of their approach. Second, they do not consider customization of views. Third, they consider unstructured process models while we focus on structured ones.

Schulz and Orlowska [26] focus on architectural support for workflow (process) views. They look into possible interactions that can occur between workflow views and between workflow views and private workflows. Next, they analyze

how such interactions can be supported by describing different ways of coupling workflow views and private workflows. Finally, they define a cross-organizational workflow architecture that supports the execution of workflow views. Our work complements their work, since we focus on how customized process can be constructed from a given business process, which is not considered in [26]. The customized process views obtained with our approach can be executed using the technology described by Schulz and Orlowska [26].

This paper builds on earlier research. In the CrossFlow project [11, 29], which developed technology to support the execution of cross-organizational workflows in dynamic virtual enterprises, already a distinction was made between external and internal level process models. There, an external level process model was specified in a contract, but no support was provided for constructing this external process model, i.e., the CrossFlow approach relies on manual construction of external process views. In follow-up research, Grefen et al. [12] defined a three-level framework for process outsourcing. In this work, also a distinction between external and internal models is made, but only abstract construction rules are presented. This paper complements that work by showing how external process views can be automatically constructed from conceptual process models. Moreover, customization of process views is not addressed in [12]. In a follow-up paper [13], Grefen et al. examined how enactment of (non-customized) process views can be supported using web service technology. A similar enactment infrastructure can be used to enact customized process views.

Some existing industrial standard, notably BPEL [3], distinguish between an abstract and a concrete process. The abstract process is a nondeterministic protocol describing possible interactions, whereas a concrete process is actually executable by a process engine. However, no consistency constraints between these notions are defined. Also, customization is not addressed.

7 Conclusion

We have presented a two-phase approach for constructing process views from structured process models. The main contribution of the approach is that it supports both providers and consumers. Providers can hide private details from their internal process models and consumers can remove noise from provider process views. The approach is formally defined, which enables an automated implementation. This allows an efficient way of constructing customized views. Since we consider structured process models, the approach fits well with the industrial process standard BPEL [3], which is mainly structured.

Next, we have shown that the approach can be architecturally supported by embedding it into an existing architecture for forming dynamic virtual enterprises. This shows how the approach can be used in architectures supporting dynamic business-to-business collaborations that are process-oriented rather than function-oriented. In today's business world, such dynamic cooperations between autonomous organizations becomes increasingly important. In the past, organizations cooperated with each other in rather static networks. To comply

with current market settings, however, organizations have to shift their priority to flexibility and ability to change if they want to survive [23]. As a consequence, dynamic cooperation between organizations is often required to meet market demands [10]. Our approach can aid in establishing such dynamic collaborations in an efficient way.

There are several directions for further work. First, we are currently implementing the aggregator and customizer module in a BPEL editor tool. Next, the approach can be extended to deal with multilateral views. These are especially useful in a choreography setting. Finally, the approach can be used to study agent-based negotiation of views between consumers and providers. For example, a provider agent may get a request from a consumer agent to show certain activities from its internal process model. The provider agent can compute a process view fulfilling this request, using the approach of this paper, and then decide on whether or net the offer is profitable enough to accept the process view.

References

1. W.M.P. van der Aalst and A. Kumar. XML Based Schema Definition for Support of Inter-organizational Workflow. *Information Systems Research*, 14(1):23–46, 2003.
2. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
3. T. Anders, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, Microsoft Corporation, SAP AG, Siebel Systems, 2002.
4. M. Carpenter, N. Mehandjiev, and I.D. Stalker. Flexible behaviours for emergent process interoperability. In S. Reddy, editor, *Proc. WETICE 2006*, pages 249–255. IEEE Computer Society, 2006.
5. I. Chebbi, S. Dustdar, and S. Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data Knowl. Eng.*, 56(2):139–173, 2006.
6. D.K.W. Chiu, S.C. Cheung, S. Till, K. Karlapalem, Q. Li, and E. Kafeza. Workflow view driven cross-organizational interoperability in a web service environment. *Inf. Tech. and Management*, 5(3-4):221–250, 2004.
7. CrossWork consortium. Crosswork project, IST no. 507590. `http://www.crosswork.info`.
8. CrossFlow. Final Report (D16). Available at `http://www.crossflow.org`.
9. R. Eshuis, P. Grefen, and S. Till. Structured service composition. In S. Dustdar, J.L. Fiadeiro, and A. Sheth, editors, *Proc. of the 4th International Conference on Business Process Management (BPM 2006)*, volume 4102 of *Lecture Notes in Computer Science*, pages 97–112. Springer, 2006.
10. P. Grefen. Towards dynamic interorganizational business process management (keynote talk). In S. Reddy, editor, *Proc. WETICE 2006*, pages 13–18. IEEE Computer Society, 2006.
11. P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig. CrossFlow: Cross-organizational Workflow Management in Dynamic Virtual Enterprises. *International Journal of Computer Systems, Science, and Engineering*, 15(5):277–290, 2001.

12. P. Grefen, H. Ludwig, and S. Angelov. A three-level framework for process and data management of complex e-services. *International Journal of Cooperative Information Systems*, 12(4):487–531, 2003.
13. P. Grefen, H. Ludwig, A. Dan, and S. Angelov. An analysis of web services support for dynamic business process outsourcing. *Information and Software Technology*, 48(11):1115–1134, 2006.
14. P. Grefen (editor) and CrossWork consortium. Global architecture (CrossWork Deliverable D4.1). 2005.
15. D. Harel, A. Pnueli, J. P. Schmidt, and S. Sherman. On the formal semantics of statecharts. In *Proceedings of the Second IEEE Symposium on Logic in Computation*, pages 54–64. IEEE, 1987.
16. D. Harel and R.E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.
17. B. Kiepuszewski, A.H.M. ter Hofstede, and C. Bussler. On structured workflow modelling. In B. Wangler and L. Bergman, editors, *Proc. CAiSE '00*, pages 431–445. Springer, 2000.
18. D.-R. Liu and M. Shen. Workflow modeling for virtual processes: an order-preserving process-view approach. *Inf. Syst*, 28(6):505–532, 2003.
19. D.-R. Liu and M. Shen. Business-to-business workflow interoperation based on process-views. *Decision Support Systems*, 38(3):399–419, 2004.
20. R. Liu and A. Kumar. An analysis and taxonomy of unstructured workflows. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proc. 3rd Conference on Business Process Management (BPM 2005)*, Lecture Notes in Computer Science 3649, pages 268–284, 2005.
21. D. Martin (editor), M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. Owl-s: Semantic markup for web services, 2004. `http://www.daml.org/services/owl-s/1.1/overview`.
22. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
23. R. Pieper, V. Kouwenhoven, and S. Hamminga. *Beyond the hype: E-business strategy in leading European companies*. Van Haren Publising, 2002.
24. A. Pnueli and M. Shalev. What is in a step: On the semantics of statecharts. In T. Ito and A.R. Meyer, editors, *Theoretical Aspects of Computer Software*, Lecture Notes in Computer Science 526, pages 244–265. Springer, 1991.
25. RosettaNet consortium. RosettaNet. `http://www.rosettanet.org`.
26. K.A. Schulz and M.E. Orlowska. Facilitating cross-organisational workflows with a workflow view approach. *Data Knowl. Eng.*, 51(1):109–147, 2004.
27. Supply Chain Council. Supply chain operations reference model (SCOR). `http://www.supply-chain.org`, 2006.
28. UML Revision Taskforce. *UML 2.0 Superstructure Specification*. Object Management Group, 2003. OMG Document Number ptc/03-07-06. Available at `http://www.uml.org`.
29. J. Vonk and P.W.P.J. Grefen. Cross-organizational transaction support for e-services in virtual enterprises. *Distributed and Parallel Databases*, 14(2):137–172, 2003.
30. X. Zhao, C. Liu, and Y. Yang. An organisational perspective on collaborative business processes. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proc. 3rd Conference on Business Process Management (BPM 2005)*, Lecture Notes in Computer Science 3649, pages 17–31, 2005.