Activity-Centric and Artifact-Centric Process Model Roundtrip

Andreas Meyer(✉) and Mathias Weske

Hasso Plattner Institute at the University of Potsdam, Potsdam, Germany
{Andreas.Meyer,Mathias.Weske}@hpi.uni-potsdam.de

Abstract. Currently, two major process modeling paradigms exist: activity-centric and artifact-centric. They focus on different first class modeling constructs and therefore, they are eligible for different scenarios. Nevertheless, both paradigms compete for users raising the own capabilities over the other's ones neglecting that both paradigms are compatible to each other such that one can transform one into the other one. In this paper, we provide a set of algorithms to allow these transformations as roundtrip, ie from an artifact-centric process model to an activity-centric one and back and vice versa. To this end, we utilize a synchronized object life cycle as mediator between both paradigms. We show applicability of our algorithms by discussing them in combination with an example.

Keywords: Process modeling · Activity-centric · Artifact-centric · Object life cycle · Model transformation

1 Introduction

Since the 1990s, workflow modeling received much attention, because of the need to specify organizations' workflows and business processes structurally and to use this representation for analysis, improvement, control, management, and enactment of the processes [1]. Today, two process modeling paradigms are of major importance: activity-centric and artifact-centric process modeling. structures (gateways) as first class modeling constructs and regards data objects in specific data states as pre- and postconditions for activity enablement or as main decision indicator at exclusive gateways. The main representative and industry standard is the Business Process Model and Notation [2]. The usage of one data object in different data states in combination with multiple activities allows to derive a so-called object life cycle, which describes the manipulations performed on a data object [3,4]. Artifact-centric process modeling [5–8] regards data objects and their object life cycles as first class modeling constructs and multiple data objects synchronize on their data state changes, ie data state changes in different object life cycles need to be performed together. The synchronization information is stored with the object life cycles instead of a control

N. Lohmann et al. (Eds.): BPM 2013 Workshops, LNBIP 171, pp. 167–181, 2014.
DOI: 10.1007/978-3-319-06257-0_14,

unit. Subsequently, the order of activities is not modeled explicitly, but can be extracted by analyzing the artifact-centric process model.

Currently, both process modeling paradigms compete for users, where especially the artifact-centric one lacks major evidence of applicability. It is proved to be useful if the process flow follows from data objects as, for instance, in manufacturing processes [7]. In contrast, in many domains, eg accounting, insurance handling, and municipal procedures, the process flow follows from activities, which need to be executed in a predefined order. While most research only considers one paradigm, we present a set of algorithms allowing the transformation of process models of one paradigm into the other one via a synchronized object life cycle–a set of object of life cycles with their transitions synchronized indicating which data state changes need to occur simultaneously. Liu et al. transform an activity-centric process model into a synchronized object life cycle utilizing the notion of data object dominance [9]. However, the authors have a restricted view on the usage of data objects, because only objects written by an activity are considered for data state determination. Implicit state transitions resulting from, for instance, underspecified process models are missing.The algorithms presented here consider paths of process models to handle underspecification properly. We also discussed this issue recently [10].

The remainder of the paper is structured as follows. Section 2 introduces four types of process models, ranging from artifact-centric to activity-centric process models before we discuss the algorithms to transform one process model type into another one by ensuring a roundtrip from one of the mentioned paradigms to the other and back. Finally, we discuss related work in Sect. 4 and conclude the paper in Sect. 5.

2 Types of Business Process Models

In the context of artifact-centric and activity-centric process models, we identified four different types of process models (cf. Definitions 2 to 5). Thereby, object life cycles describe the allowed data object manipulations and the interdependencies of multiple data objects in the course of process model execution. These can be combined in a synchronized object life cycle, which acts as mediator between activity- and artifact-centric process models as both require to align to this data flow specification. Further, an activity-centric process model may exist in two variations: with and without information about attribute types. Below, we introduce each process model type. But first, we start with the definition of a data object and data object class. Both concepts are the major connection between the artifact-centric and the activity-centric modeling paradigm.

Definition 1 (Data object and data object class). A *data object class* $C = (\mathcal{P}, S, i, S_F)$ consists of a finite set $\mathcal{P}$ of attributes, a finite non-empty set S of data states ($\mathcal{P}, S$ are disjoint), an initial data state $i \in S$, and a non-empty set $S_F \subseteq S$ of final data states. A *data object* $D = (s, P)$ consists of a data state $s \in S$ and a finite set $P \subseteq \mathcal{P}$ of attributes. Each data object D is an instance of

a data object class C such that $\varphi(D) = C$, where $\varphi : \mathcal{D} \rightarrow \mathcal{C}$ with $\mathcal{D}$ denoting the finite set of all data objects and $\mathcal{C}$ denoting the finite set of all data object classes. ◇

The concept of business artifacts [6,11] has been established based on the initial work of Nigam and Caswell [5]. We define an artifact-centric process model – closely related to existing work [11] – as follows.

Definition 2 (Artifact-centric process model). An *artifact-centric process model* $ACP = (Z, V, B)$ consists of a schema $Z = (\mathcal{C}, \lambda, \kappa)$ with a finite non-empty set of data object classes $\mathcal{C}$, the *in-state* function $\lambda : \mathcal{C} \times S \rightarrow \{true, false\}$, and the *defined* function $\kappa : \mathcal{C} \times P \rightarrow \{true, false\}$, a finite set V of tasks, and a finite set B of business rules. Functions λ and κ evaluate to true or false depending on the existence of a data object of class C being in a data state s or containing a value for the attribute p respectively. A task $v = (label, O)$, $v \in V$, consists of a label and a finite set $O \in C$ of data object classes referring to data objects being manipulated by this task. A business rule $b = (pre, post, W)$ consists of a precondition *pre*, a postcondition *post*, and a finite set $W \subseteq V$ of tasks manipulating data objects to meet the postcondition. A pre- as well as a postcondition comprises a set of in-state and defined functions connected by operators $\wedge$ and $\vee$. Thereby, a defined function may only contain a data object class, which is used in at least one in-state function. ◇

Artifact-centric process models focus on the data objects involved in the process with each object having a life cycle and being synchronized by their state transitions. Following, a commonly used visual representation of such artifact-centric process model is a synchronized object life cycle, which also describes the execution semantics of such process model. We define a synchronized object life cycle as follows.

Definition 3 (Synchronized object life cycle). A *synchronized object life cycle* $\mathfrak{L} = (\mathcal{L}, SE)$ consists of a finite set $\mathcal{L}$ of object life cycles and a finite set SE of synchronization edges. An object life cycle $L = (S, i, S_F, T, \Sigma, \eta)$, $L \in \mathcal{L}$, consists of a finite set S of data states, an initial data state $i \in S$, a non-empty set $S_F \subseteq S$ of final data states, a finite set $T \subseteq S \times S$ of data state transitions, and a finite set Σ of actions representing the manipulations on data objects. Function $\eta : T \rightarrow \Sigma$ assigns an action to each data state transition. Each object life cycle L corresponds to a data object class C such that $\psi(L) = C$, where $\psi : \mathcal{L} \rightarrow \mathcal{C}$ with $\mathcal{C}$ denoting the finite set of all data object classes. A synchronization edge $e = (t_1, t_2)$, $e \in SE$, connects two object life cycles $L_1, L_2 \in \mathcal{L}$ by assigning an edge between data state transitions $t_1 \in T_{L_1}$ and $t_2 \in T_{L_2}$ being part of object life cycles L_1 and L_2 respectively. ◇

For a data state transition from state $s1$ to state $s2$ we refer to $s1$ as source and to $s2$ as target. The second process modeling paradigm focuses on the partial order of activities, but also requires data flow information for execution, which is comprised as second class modeling artifact. Therefore, the correct usage of data

objects needs to be verified by checking them against the allowed manipulations specified in the corresponding object life cycles. We define such activity-centric process model as follows.

Definition 4 (Activity-centric process model). An *activity-centric process model* $M = (N, \mathcal{D}, Q, C, F, type, \mu, \xi)$ consists of a finite non-empty set $N \subseteq A \cup G \cup E$ of nodes being activities A, gateways G, and Events $E \subseteq E_S \cup E_E$ comprising start and end events, a finite non-empty set $\mathcal{D}$ of data objects, and a finite non-empty set Q of activity labels ($N, \mathcal{D}, Q$ are pairwise disjoint). The set of attributes P of a data object $D \in \mathcal{D}$ is suppressed in this type of process model such that $P = \emptyset$. $C \subseteq N \times N$ is the control flow relation, $F \subseteq (\mathcal{D} \times A) \times (A \times \mathcal{D})$ is the data flow relation representing read respectively write operations of activities with respect to data objects; $type : G \rightarrow \{xor, and\}$ assigns to each gateway a type, $\mu : A \rightarrow Q$ assigns to each activity a label, and $\xi : (G \times (A \times G) \nrightarrow D$ assigns data conditions to control flow edges having an xor gateway as source. $\diamond$

In this paper, we refer to read data objects as *input* and to written data objects as *output* data objects. We call a gateway $g \in G$ an *xor* (*and*) gateway if $type(g) = xor$ ($type(g) = and$). An xor (and) gateway with two or more outgoing edges is called split (fork) and an xor (and) gateway with two or more incoming edges is called join (merge). As usual, we assume process model M to be structural sound, i.e., M contains exactly one start and one end event and every node of M is on a path from the start to the end event. Further, each activity of M has exactly one incoming and one outgoing control flow edge. Finally, we define an activity-centric process model with attribute definitions as extension to M as follows.

Definition 5 (Activity-centric process model with attribute definition). An *activity-centric process model with attribute definition MX* is an activity-centric process model M, where the set of attributes P of a data object D is regarded. $\diamond$

For all notions, we use subscripts, e.g., C_{ACP}, S_L, A_M, and D_{XM}, to denote the relation of the sets and functions introduced above to one of the process models. Subscripts are omitted where the context is clear. Attributes are considered to be flat for all process model types, ie attribute nesting is not in scope of this paper.

3 Roundtrip

In this section, we introduce five algorithms to transform the different types of process models defined in Sect. 2 into each other. Figure 1 shows allowed transformations and aligns the corresponding algorithm to

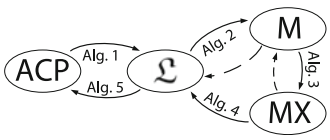

Fig. 1. Transformations between different types of process models

Algorithm 1. Transformation of an artifact-centric process model $ACP = (Z, V, B)$ into a synchronized object life cycle $\mathfrak{L} = (\mathcal{L}, SE)$

1: create object life cycle for each data object class
2: for each business rule, add (if not existing yet) states used as precondition respectively postcondition and
corresponding transitions labeled with the task name of the business rule to the respecting object life cycle
3: add synchronization edges between transitions of different object life cycles extracted from the same business rule
4: identify initial and final state in each object life cycle

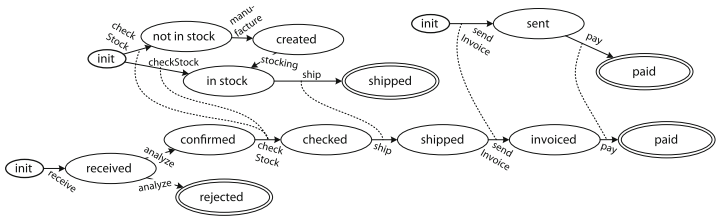

Fig. 2. Synchronized object life cycle

each of them. Algorithm 1 transforms an artifact-centric process model ACP into a synchronized object life cycle $\mathfrak{L}$, which in turn can be transformed into an activity-centric process model M by using Algorithm 2. The enrichment of a process model M with data object attribute information towards MX is described in Algorithm 3. Required information about the attributes is derived from the respecting ACP. For the opposite transformation, we do not provide an explicit algorithm, because it suffices to remove the attribute information to derive M from MX. Further, Algorithm 4 allows the transformation of a process model MX with attribute definition into a synchronized object life cycle $\mathfrak{L}$. It can be easily adapted to use M as its basis. Finally, Algorithm 5 transforms a synchronized object life cycle $\mathfrak{L}$ into an artifact-centric process model ACP by considering information about attributes taken from MX. All algorithms will be explained in detail and supported by brief algorithm representations; fully detailed algorithm representations are provided in our technical report [12].

We start with Algorithm 1. First, the single object life cycles to be comprised by the synchronized one are initialized, one for each data object class used in the business rules of the respecting artifact-centric process model (line 1). Then, we analyze the business rules to extract the data object states and the transitions and dependencies between them. For each business rule, first, the data states used as precondition or postcondition are extracted and added to the set of states of the corresponding data object class if they are not yet present in the respecting set of states. Next, for each data object class, we add a transition from each state used in the precondition to each state used in the postcondition where both states belong to that data object class. Afterwards, the tasks of each business rule are extracted and the tasks labels are assigned to priorly

added data state transitions. Thereby, multiple tasks used in one business rule are combined such that their labels are concatenated with + as operator due to the semantics of activity-centric process models, where all activities changing the input data objects into the output data objects are comprised by one single activity instead of multiple tasks affecting different state transitions as allowed for artifact-centric process models (line 2). Finally, after identifying all data state transitions for all data object classes used in one business rule, we synchronize the different object life cycles with respect to these data states by adding a synchronization edge between each two transitions belonging to different object life cycles indicating that these are executed together (line 3). For ensuring the alignment of each object life cycle to definition 3, the state without an incoming transition becomes the initial state of the specific object life cycle and the all states without an outgoing transition become a final data state (line 4).

Table 1. Artifact-centric process model

Data object classes:	order, product, invoice
Set of tasks:	receive, analyze, checkStock, manufacture, stocking, ship, sendInvoice, receivePayment, setPayed
Business rules:	b1, b2, b3, b4, b5, b6, b7, b8
b1: Organization receives order from customer	
Precondition:	$\lambda(\text{Order}, \text{init})$
Tasks:	receive(Order)
Postcondition:	$\lambda(\text{Order}, \text{received}) \wedge \kappa(\text{Order}, \text{CustomerNumber}) \wedge \kappa(\text{Order}, \text{ReceiveDate}) \wedge \kappa(\text{Order}, \text{Products})$
...	...
b3: Organization checks warehouse stock for product availability	
Precondition:	$\lambda(\text{Order}, \text{confirmed}) \wedge \lambda(\text{Product}, \text{init})$
Tasks:	checkStock(Order, Product)
Postcondition:	$(\lambda(\text{Product}, \text{inStock}) \vee \lambda(\text{Product}, \text{notInStock})) \wedge \lambda(\text{Order}, \text{checked})$
...	...
b8: Organiazation receives payment for order from customer	
Precondition:	$\lambda(\text{Order}, \text{invoiced}) \wedge \lambda(\text{Invoice}, \text{sent}) \wedge \kappa(\text{Invoice}, \text{Amount})$
Tasks:	receivePayment(Invoice) setPayed(Order)
Postcondition:	$\lambda(\text{Order}, \text{paid}) \wedge \lambda(\text{Invoice}, \text{paid}) \wedge \kappa(\text{Order}, \text{PaymentDate})$

Table 1 shows an extract of an artifact-centric process model describing a simple order and delivery process consisting of three data object classes, nine tasks, and eight business rules, where three of them are completely presented. Due to space requirements, the others are omitted. Based on Algorithm 1, this process model can be mapped into the synchronized object life cycle shown in Fig. 2 consisting of three object life cycles corresponding to the data object classes. Based on business rule *b3*, transitions from data state *init* to *in stock* respectively *not in stock* labeled with the *checkStock* are added to the life cycle of data object *Product* and a transition from *confirmed* to *checked* also labeled *checkStock* is added for the life cycle of *Order*. Additionally, both transitions of the product life cycle are synchronized with the one from the order life cycle.

In Algorithm 2, the transformation of a synchronized object life cycle into an activity-centric process model without attribute definitions (cf. definition 4), the first step is the identification of data state transitions, which are executed together. Therefore, all transitions with the same label are grouped into a combined transition; more specifically, a combined transition comprises the transitive closure over all transitions of the synchronized object life cycle being connected by a synchronization edge. Next, the activity-centric process model is created with a start event as only node (lines 1 to 2).

Then, we iteratively create the process model until no more nodes can be extracted from the synchronized object life cycle (lines 3 to 8). With respect to the nodes of the process model, we distinguish whether it has already been checked for succeeding nodes or not. Each node needs to be checked exactly once. Therefore, in each iteration, we start with the nodes that have not been checked yet, i.e., the nodes added to the process model in the previous iteration. For each such not yet checked node n, we derive the set of combined transitions of the synchronized object life cycle, which are enabled after termination of that node. Additionally, we derive the set of all combined transitions, which might be enabled after termination of that node. Therefore, we determine the combined transition and the corresponding transitions t executed by node n and collect all combined transitions of the synchronized object life cycle, which contain a transition having the target state of t as source state. We call these combined transitions *potentially enabled transitions*. For each such potentially enabled transition, we check whether it becomes enabled after termination of all enabled nodes. If not, we remove that combined transition from the collection. If the node contains output data objects reaching the final state of the corresponding object life cycle, we add an activity labeled $nop : state$ to the process model, where *state* refers to the actual final data state. This activity does not contain any data association. Additionally, a corresponding combined transition is added to the set NOP_n of no operation activities of the specific node n such that this activity gets directly marked as checked for combined transitions. Before checking the next node, the currently one is marked as checked for combined transitions (line 4).

Next, we create for each enabled or potentially enabled combined transition an activity and define the corresponding data object accesses, where the source states of all transitions grouped in the respecting combined transition are read while the target states are written. Third, the activity gets assigned the action names of the transitions as activity label. Thereby, the names of multiple transitions are concatenated by using the +. In post processing, a stakeholder may adapt the activity labels (line 5).

Next, we establish the control flow of the activity-centric process model first focusing on sequences, splits, and forks before we focus on joins and merges. If, for a node, the number of combined transitions being enabled equals one, we add a control flow edge from that node to the activity created for the combined transition relating to the node. If the number exceeds one, we either add a split if the set NOP_n of no operation activities of node n is empty or we intersect the combined transitions to determine the split respectively fork for cases where

Algorithm 2. Transformation of a synchronized object life cycle $\mathfrak{L} = (\mathcal{L}, SE)$ into an activity-centric process model $M = (N, \mathcal{D}, Q, C, F, type, \mu, \xi)$

1: group transitions executed together, i.e., identically named ones, into combined transitions
2: create activity-centric process model with a single start event
3: **repeat**
4: identify enabled combined transitions for all not yet checked nodes of the process model
5: for each identified combined transition, create an activity with input data objects conforming to the source states and output data objects conforming to the target states of the transitions comprised in the respecting combined transition; the label corresponds to one of the transition actions
6: add control flow edges to process model (if there exist several combined transitions for one node, an xor gateway with respecting edge conditions respectively an and gateway is added as well)
7: for activities with more than one incoming control flow edge, an xor respectively and gateway is added preceding that activity and the control flow is rerouted accordingly
8: **until** all nodes have been checked for enabled combined transitions
9: combine all paths of the process model with an xor gateway (if necessary) and route them towards a single end event also added to the process model

NOP_n is empty. If the intersection reveals at least one data object class being used in all combined transitions, we create an xor gateway (split), otherwise we create an and gateway (fork). Then, control flow edges from the node to the created gateway as well as from the gateway to each activity being created for respecting combined transitions are added to the process model. For splits, we also add the edge conditions to the corresponding edges by assigning the data object being input to the activity the edge targets to and being output of the activity being the predecessor of the edge's source, the split. The added gateway gets marked as checked for combined transitions as we processed all paths with above described actions (line 6). Afterwards, all activities having more than one incoming control flow edge are adapted by adding an xor respectively an and gateway, which consumes all these control flow edges. Further, a control flow edge connects that gateway with the activity. To distinguish the type of gateway, we retrieve the combined transition for each activity being the first preceding one of the activity with multiple incoming control flow edges. We intersect these combined transitions as described above. Data object classes used in all combined transitions indicate an xor gateway, otherwise we set the gateway type to and. Again, the added gateway is marked as checked for combined transitions (line 7).

We finalize the process model by adding a single end event. If there exists exactly one node, or more specifically one activity, without an outgoing control flow edge, either this node is connected to the end event via a control flow edge, if it is commonly labeled, or the control flow edge targeting the node is rerouted towards the end event and the node is removed from the process model, if it is labeled with $nop : state$. In case there exist multiple nodes without an outgoing control flow edge, an xor gateway is also added, which connects to the end event. Each such node being labeled with $nop : state$, the control flow edge targeting the node is rerouted to target the added xor gateway while the node gets removed

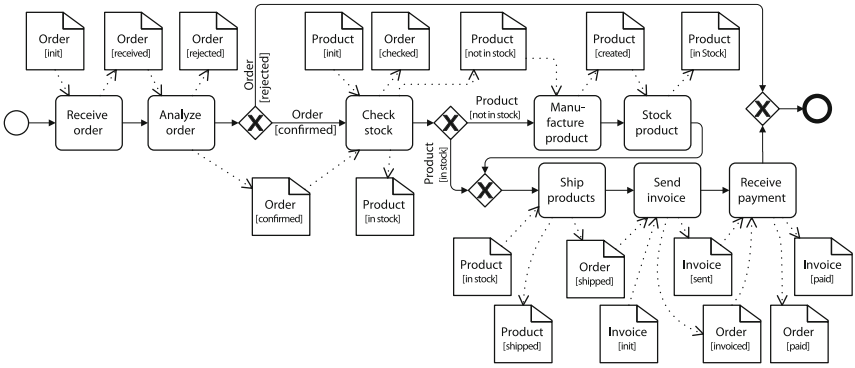

Fig. 3. Activity-centric process model

Algorithm 3. Transformation of an activity-centric process model $M = (N, \mathcal{D}, Q, C, F, type, \mu, \xi)$ into an activity-centric one MX with attribute definition

1: extract XML representation of activity-centric process model
2: **for all** data objects of the activity-centric process model **do**
3: extract attribute types from business rules of corresponding artifact-centric process model
4: extend XML representation of the data object accordingly to Listing 1
5: **end for**

from the process model. For all other such nodes, the node is connected to the xor gateway via a control flow edge (line 9).

Figure 3 shows the activity-centric process model resulting from the synchronized object life cycle given in Fig. 2 after applying Algorithm 2. After termination of activity *Analyze order*, the combined transition comprising all transitions labeled *checkStock* and the one representing a no operation path for the *reject* case are enabled. As there are two combined transitions enabled with one being a no operation path, we add an xor gateway and control flow edges to the corresponding activities *nop:reject* and *Check stock*. The latter one has two input (*Order* in state *confirmed* and *Product* in state *init*) and three output data objects (*Order* in state *checked* and *Product* in states *in stock* and *not in stock* respectively). The *nop:reject* activity has no associated data objects and gets removed while integrating all paths into the single end event.

The first step towards the activity-centric process model with attribute definition is to extract the XML representation from the activity-centric process model as noted in line 1 of Algorithm 3. This extraction aligns to standard XML extractions as, for instance, aligning to the one described in the BPMN specification [2]. Afterwards, all data objects used in the process model are determined (line 2). For each data object, the attribute types are extracted from the business rule of the respecting artifact-centric process model, because the attribute information is only available there (line 3). The correct business rule is identified via the data object and its state as well as the type of access, i.e., read or write.

A read data object is aligned to the precondition and a written data object is aligned to the postcondition of the business rule containing the respecting data state. Based on the defined functions specified in the pre- respectively postcondition, the XML representation of the data object is extended with attribute tags using extension mechanisms (line 4). For BPMN, the extension mechanism is called extension points and allows to specify for each tag *extension elements*. Listing 1 shows the structure of the XML representation of an extended data object. The extension comprises lines 2 to 7 and is adapted to the number of attributes extracted from the business rule. The attribute type is added to the *name* field of the *attribute* tag; the value is set in the course of process execution. Generally, an empty value indicates that an attribute is not defined while an existing value indicates successful attribute definition.

Based on business rule *b1* from Table 1, data object *Order* being output to activity *Receive order* in Fig. 3 gets extended with the attributes *CustomerNumber*, *ReceiveDate*, and *Products*; on model level, each has an empty value.

```
1 <dataObject id="" name="">
2   <extensionElements>
3     <attribute id="" name="">value</attribute>
4     ...
5   </extensionElements>
6   <dataState id="" name="" />
7 </dataObject>
```

Listing 1. Extended XML representation of data object comprising attribute information

Algorithm 4. Transformation of an activity-centric process model MX with attribute definition into a synchronized object life cycle $\mathfrak{L} = (\mathcal{L}, SE)$

```
1: create object life cycle for each data object class
2: for all traces through the activity-centric process model with attribute definition do
3:   repeat
4:     if node is an activity then
5:       add input data states of node to corresponding object life cycle and connect them via
         τ-labeled
         transitions to respecting previous states
6:       synchronize transitions belonging to different object life cycles
7:       add output data states of node to corresponding object life cycle and connect them
         via transitions to respecting previous states considering succeeding xor blocks; the
         label of the node is mapped to the action of the transition
8:       synchronize transitions belonging to different object life cycles
9:     end if
10:  until trace has no next node
11: end for
12: identify initial and final state in each object life cycle
```

The extraction of a synchronized object life cycle from an activity-centric process model is described in Algorithm 4. First, all data object classes used in the activity-centric process model are identified before for each of them, the corresponding object life cycle consisting of the initial state only is created and the

distinct data states are determined. Both information is stored in the corresponding sets of data object classes and data states; the latter one exists separately for each class (line 1). Next, we extract all traces from the start event to the end event of the process model (loops are reduced to a single trace). Then, we handle each trace separately (lines 2 to 11). Thereby, we first create the object life cycle specific collections K_C, which will be used to store data states relating to data objects of the corresponding class. The initial state of the object life cycle is added to each collection. Then, all nodes of the trace are checked for their type and processed accordingly (line 4).

If the node is an activity, the set of input data objects is received grouped by data object classes. Afterwards, if not existing yet, the identified input data states are added to the corresponding object life cycle and the transitions between them are specified; from each entry of the data state collection K_C to each data state identified for the data object class, one transition is added to the object life cycle, if source and target are different states. Each of these transitions gets assigned τ as action, which requires adaptation from the stakeholder in post processing, because these transitions cover implicit data state transitions of the process model. The last step in this regard is to replace the content of collections K_C with the data states valid for the current node (line 5). Adding a synchronization edge between each two transitions just added and belonging to different object life cycles (representing a combined transition) finalizes the processing of the input data objects of the current node (line 6). Next, we process the output data states of the node again grouped by data object classes. After extracting the output data states, they need to be filtered whether they are used as edge condition on outgoing control flow edges of an succeeding split such that not all outgoing data states are utilized in this trace. If a data object class is part of such edge condition, all data states not matching the condition are removed from the set of identified output states. The remaining data states are added to the set of states of the corresponding object life cycle and the new transitions are specified. Comparably to the input data objects, one transition is added from each data state currently stored in data state collection K_C to each remaining output data state independently from source and target state. Following, differently to the input data objects, self-loops are allowed. Afterwards, each newly added transition gets assigned the activity label as action. Transitions being skipped for addition due to existence get their action extended by the label of the current node. Again, the action labeling can be adapted by the stakeholder in post processing (line 7). The output state processing closes with the replacement of data states as discussed for the input data objects before synchronization edges are added between each two transitions being added or skipped above and belonging to different object life cycles (line 8).

If the node is a split, the collection of data states for the data object class used in the edge condition on the outgoing control flow edges is adapted. It is set to the state of the data object used in this trace. After processing all traces, we finalize Algorithm 4 by specifying each state without an incoming transition

Algorithm 5. Transformation of a synchronized object life cycle $\mathfrak{L} = (\mathcal{L}, SE)$ into an artifact-centric process model $ACP = (Z, V, B)$

1: group transitions executed together, i.e., identically named ones, into combined transitions
2: for each object life cycle, create data object class with respecting sets of states and final states and the initial state
3: for each data object, extract the attributes from the activity-centric process model with attribute definition
4: create business rules with information from combined transitions (tasks, in-state functions) and the attributes (defined functions)
5: build artifact-centric process model with a schema comprising all data object classes, the business rules, and the set of tasks utilized in the business rules

as initial and each state without an outgoing transition as final state of the respecting object life cycle (line 12).

Activity *Check stock* from the activity-centric process model with attribute definition (visual representation in Fig. 3) adds the states *init*, *in stock*, and *not in stock* with transitions from *init* to the other two states to the *Product* life cycle and adds state *checked* with a transition from *confirmed* to *checked* to the *Order* life cycle. The latter transition is synchronized with the others. All transitions get assigned the action *checkStock*.

Finally, the transformation of an synchronized object life cycle into an artifact-centric process model is described in Algorithm 5. This transformation requires information about the attributes of the data objects, which are taken from the corresponding activity-centric process model with attribute definition. Similarly to Algorithm 2, we first group the transitions of the synchronized object life cycle into combined transitions. Two transitions are executed together if they are connected by a synchronization edge (line 1). Next, for each object life cycle of the synchronized one, we create a corresponding data object class, which gets the states, the initial state, and the final states from that object life cycle. Additionally, for each object life cycle, we create an empty K_C map, which stores for each data state concatenated with a unique identifier a collection of defined functions (line 2). Then, we extract the attribute information from the activity-centric process model with attribute definition. Thereby, we parse the XML structure and for each data object, the set of attribute types is extracted and added to map K_C as well as the set of attributes of the respecting class (line 3). Afterwards, the combined transitions that have been determined in beginning are processed. For each combined transition, one business rule is created. The task affected by the business rule is derived from the action of the transitions (they are equal for all transitions comprised by one combined transition) and added to the business rule. Then, we derive the in-state and defined functions for both the pre- and postcondition of the business rule. An in-state function consists of the data object class the respecting transition belongs to and the source respectively target data state of that transition. The defined functions are taken from map K_C of the data object class the respecting transition belongs to depending on the source respectively target data state of that transition. The assignment of these functions to the pre- and postcondition of the business rule requires

a grouping of them with respect to both conditions as well as the corresponding data object class. The elements within one group are connected by the $\vee$ operator while different groups belonging to the same condition are connected by the $\wedge$ operator. Based on the condition, these two statements are added to the pre- respectively postcondition of the business rule (line 4). Finally, the actual artifact-centric process model consisting of a schema comprising all data object classes, a set of business rules, and a set of tasks utilized in the business rules is created (line 5).

Transformation of the synchronized object life cycle presented in Fig. 2 to the artifact-centric process model partly shown in Table 1 is achieved by applying Algorithm 5. Considering the combined transition comprising transitions with label *paid*, a new business rule is created (cf. *b8*) with the source states being added to the precondition and the target states being added to the postcondition for both affected data object classes. Differently to the representation in Table 1, the corresponding task is summarized in one task with both classes as arguments: *pay(Order, Invoice)*. $\kappa(Order, PaymentDate)$ is extracted from the corresponding data object of the activity-centric process model with attribute definition. The data object classes are *order*, *product*, and *invoice*.

For both algorithms with required external information, we utilize this information if it is present. Otherwise, the algorithms ignore the corresponding parts resulting in incomplete process models. Algorithm 3 results in a plain XML representation of the input activity-centric process model, ie for all data objects, there do not exist any attribute tags. Algorithm 5 results in an artifact-centric process model, where the set of attributes is empty for all data object classes. Subsequently, no defined function exists in any business rule for resulting the process model.

4 Related Work

Activity-centric business process modeling emerged from workflow modeling and is described extensively in several works, e.g., [1], with BPMN being the widely used industry standard [2]. The artifact-centric modeling paradigm was initiated by IBM research [5] and further formalized by several researchers [6,11]. Additionally, deviations of this paradigm have been developed and process engines executing such process models have been established [7,8]. Both paradigms compete each other although they only provide different views on the same business processes putting either activities or data objects in focus. Instead of keeping both paradigms separated, we combined them with a set of transformation algorithms. First steps towards an integration have been taken by extracting unsynchronized [3,4] and synchronized [9] object life cycles from activity-centric process models as well as by extracting activities statically from processing paths through data objects [13] or goals [14] or dynamically from data dependencies on missing data [15]. All mentioned approaches provide means to partly support one of the five transformations introduced in this paper. Liu et al. [9] provide an approach closely related to Algorithm 4 without attribute consideration. Additionally, they also assume that each data object written in a specific state is also

read in this state, if the object gets read again. Besides considering data object attributes, we also allow the read of a previously written object, if there exists a path between the respecting states in the object life cycle (cf. [10], where we discussed the issue of underspecification).

5 Conclusion

We presented a set of five algorithms allowing to transform artifact-centric process models into activity-centric ones and vice versa via an synchronized object life cycle acting as mediator between both process modeling paradigms. We showed applicability of these algorithms by applying them to a simple order and delivery process. In future work, we will implement the algorithms to allow interested stakeholders to transform their processes from one paradigm into another; e.g., transforming an artifact-centric process model into an activity-centric one to apply existing analysis techniques.

References

1. Weske, M.: Business Process Management: Concepts, Languages, Architectures, 2nd edn. Springer, Berlin (2012)
2. OMG: Business Process Model and Notation (BPMN), Version 2.0 (2011)
3. Ryndina, K., Küster, J.M., Gall, H.C.: Consistency of business process models and object life cycles. In: Kühne, T. (ed.) MoDELS 2006. LNCS, vol. 4364, pp. 80–90. Springer, Heidelberg (2007)
4. Eshuis, R., Van Gorp, P.: Synthesizing object life cycles from business process models. In: Atzeni, P., Cheung, D., Ram, S. (eds.) ER 2012 Main Conference 2012. LNCS, vol. 7532, pp. 307–320. Springer, Heidelberg (2012)
5. Nigam, A., Caswell, N.S.: Business artifacts: an approach to operational specification. IBM Syst. J. **42**(3), 428–445 (2003)
6. Cohn, D., Hull, R.: Business artifacts: a data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull. **32**(3), 3–9 (2009)
7. Müller, D., Reichert, M., Herbst, J.: Data-driven modeling and coordination of large process structures. In: Meersman, R., Tari, Z. (eds.) OTM 2007, Part I. LNCS, vol. 4803, pp. 131–149. Springer, Heidelberg (2007)
8. Künzle, V., Reichert, M.: PHILharmonicFlows: towards a framework for object-aware process management. J. Softw. Maint. **23**(4), 205–244 (2011)
9. Liu, R., Wu, F.Y., Kumaran, S.: Transforming activity-centric business process models into information-centric models for soa solutions. J. Database Manag. **21**(4), 14–34 (2010)
10. Meyer, A., Polyvyanyy, A., Weske, M.: Weak conformance of process models with respect to data objects. In: Services and their Composition (ZEUS) (2012)
11. Yongchareon, S., Liu, Ch., Zhao, X.: A framework for behavior-consistent specialization of artifact-centric business processes. In: Barros, A., Gal, A., Kindler, E. (eds.) BPM 2012. LNCS, vol. 7481, pp. 285–301. Springer, Heidelberg (2012)
12. Meyer, A., Weske, M.: Activity-centric and artifact-centric process model roundtrip. Hasso Plattner Institute at the University of Potsdam, Technical report (2013)

13. Wang, J., Kumar, A.: A framework for document-driven workflow systems. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 285–301. Springer, Heidelberg (2005)
14. Vanderfeesten, I., Reijers, H.A., van der Aalst, W.M.P.: Product-based workflow support. Inf. Syst. **36**(2), 517–535 (2011)
15. van der Aalst, W.M.P., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data Knowl. Eng. **53**(2), 129–162 (2005)