

A Process View Framework for Artifact-Centric Business Processes

Sira Yongchareon and Chengfei Liu

Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, Victoria, Australia
{syongchareon, cliu}@swin.edu.au

Abstract. Over the past several years, the artifact-centric approach to workflow has emerged as a new paradigm of business process modelling. It provides a robust structure of workflow and supports the flexibility of workflow enactment and evolution especially in a collaborative environment. To facilitate and foster business collaborations, the customisation, privacy protection, and authority control of business processes are essential. Given the diverse requirements of different roles involved in business processes, providing various views with adequate process information is critical to effective business process management. Several approaches have been proposed to construct views for traditional process-centric business processes; however, no approach has been developed for artifact-centric processes. The declarative manner of process modelling in artifact-centric approaches makes view construction challenging. In this paper, we propose a novel process view framework for artifact-centric business processes. The framework consists of artifact-centric process models, view models, and a mechanism to derive views from underlying process models. Consistency rules are also defined to preserve the consistency between a constructed view and its base process model.

1 Introduction

The artifact-centric business process model has emerged as a new promising approach for modeling business processes, as it provides a highly flexible solution to capture operational specifications of business processes. It particularly focuses on describing the data of business processes, known as “*artifacts*”, by characterizing business-relevant data objects, their lifecycles, and related services. The artifact-centric process modelling approach fosters the automation of the business operations and supports the flexibility of the workflow enactment and evolution [1, 7, 10, 11]. Further, it also effectively enables collaboration of the business processes as the lifecycle of an artifact may span multiple business units when it evolves through the process. Each time the artifact moves from one state to another state, its relevant information/data is updated. Services that update on the artifact can be performed by various roles of participants, which may belong to a single organization or different organizations. Given the diversity of the participants involved in business processes, providing various views with adequate information of artifacts and business processes to participants according to

their roles is considered as a critical requirement to enable the effective business process modelling and management [5].

For example, top-level executives may need high level information about all important artifacts, while operational level employees may need to know the detailed information of some artifacts and their progresses. Based on these requirements, there is a need of a flexible and customizable process model to support appropriate artifact-centric process abstractions that encapsulate sensitive information for different roles of participants.

The process view approaches have been proposed in several works [2, 3, 5, 6] to provide various levels of visibility to process information which enable organizations to maintain appropriate levels of privacy and security. These works are based on the traditional process-centric business processes. To the best of our knowledge, the problem of constructing views for the artifact-centric processes has not yet been well studied. We observed that the approaches to process view construction of those two paradigms of business process modelling are different. (1) The approach to view construction of the process-centric processes is task-based abstraction, while for the artifact-centric processes is object-based. The former is more explicit while the latter is more implicit. (2) The process logic of process-centric processes is described in a procedural manner using control flows, while the logic of artifact-centric processes is captured and maintained in a declarative manner using business rules and services to define the interaction of artifacts, tasks and their flow. As such, this brings in a new challenge on how views can be defined and derived for different roles for such declaratively defined specifications of business processes.

To address the above requirements, we propose a novel artifact-centric process view framework which consists of an artifact-centric business process model, a process view model with the construction approach, and a comprehensive set of view consistency rules to preserve the structural and behavioural consistency between process views. Our proposed framework enhances the artifact-centric business process model as it can provide various roles of participants with different views of a process to meet their requirements and to support different levels of authority when accessing artifacts of collaborative business processes.

The remainder of this paper is organized as follows. Section 2 introduces process view framework with our motivating example. Section 3 introduces the artifact-centric process model, definitions, and syntaxes for both business processes and views. Section 4 provides a process view construction framework based on role-based view configurations and consistency constraints. Section 5 reviews the related works. Finally, the conclusion and future work are given in Section 6.

2 Motivating Example

In the artifact-centric approach, business processes are structured around business artifacts. An artifact holds its data and its current state. The move of one state to another state of an artifact is triggered by a pre-defined business rule of the processes, i.e., a business rule describes how state changes. This rule also induces a service invocation which will modify data of related artifacts. The detail of artifact-centric process model is described in Section 3.

In this section, we introduce an example of business processes to illustrate and motivate the process views that can be constructed based on the underlying business process with role-based view requirements. Our example of business processes is adapted from a general buyer-seller business and its supply chain process. It consists of two business processes: product ordering and shipping. The ordering process starts when a customer places an order to the retailer for a particular product and ends when the customer pays the invoice. The shipping process starts when the retailer creates a shipment and ends when the item arrives to the customer. Now, we start with introducing artifacts used in the processes. Figure 1 shows the cross-boundary view of five main artifacts that involve in our collaborative business processes. The shaded boundary of each artifact indicates that the state/data of artifact is changed or updated by some participants in the collaboration. The boundary of an artifact may cover multiple departments in the organization or even span to other organizations. Here, we do not detail how data/state of each artifact can be changed; however, some artifacts are described when we run through Section 3.

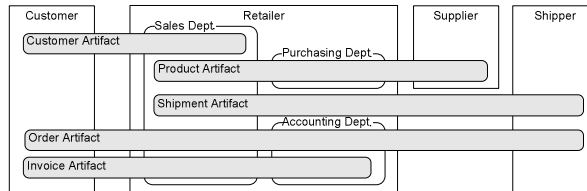


Fig. 1. Business artifacts in collaborative business processes

Now, we can see that an artifact which spans multiple departments or organizations is an interest of some participants that belong to those organizational units, e.g., the *Order* artifact involved in the Sale and Accounting departments is of course interesting to sale clerks, sale managers, accounting clerks, and accounting managers, etc. Each role of stakeholders who are interested in the artifact may have different requirements or restrictions of how much details of the artifact they want to see or are able to see depending on the authority of the role together with the privacy and security concerns for the artifact in the processes. When process modelers discover artifacts and model processes, they may not need to concern with these requirements of the stakeholders. The artifact then is defined at the most detailed level that is adequate enough to be used for the operation of processes. Later, they define/customize different views of the artifact for different roles depending on user's requirements, e.g., customization, privacy protection, and authority control. The view requirement for users of a particular role specifies a structure of artifact's states and their visibility.

Figure 2 shows different views of the *Order* artifact in our business processes. Figure 2 (a) presents the original (or operational) view of the *Order* artifact. Figure 2 (b) illustrates the views of the *Order* artifact for the *Sale* and *Accounting* roles, respectively, according to the view requirements specified in Figure 2 (c). In this example, the requirement for the *Sale* role specifies that: (1) the *delivering* and *billed* states are nested under the *in_processing* state and *delivering* nests the *in_shipping* and *shipped*; (2) the *init*, *open_for_item*, *in_shipping*, and *shipped* states are hidden. We use a white rectangle to represent a visible state while a gray shaded rectangle represents a hidden state.

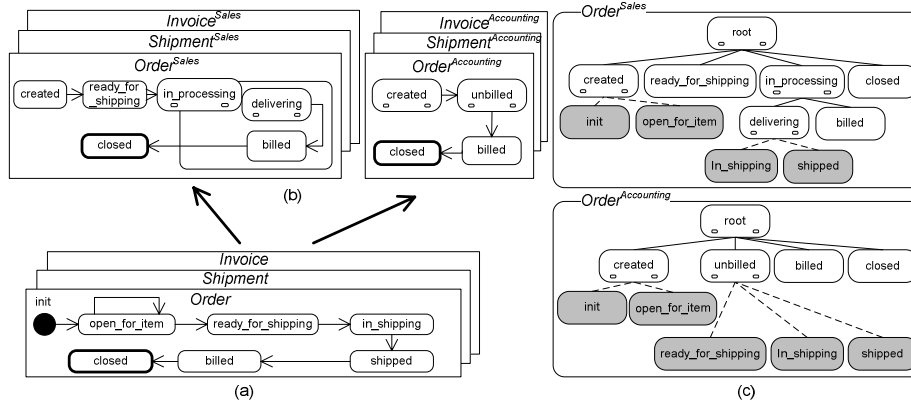


Fig. 2. Operational view vs. Role-based view

It is understandable that the view requirements can be specified at the artifact level in the artifact-centric approach, i.e., each artifact in business processes will have its corresponding requirement for each role. However, when constructing process views, we do not only consider individual artifacts, but we need to take into account business rules as well. For example, if a business rule makes a change of two states of different artifacts, then hiding a state of one artifact will have an impact on the another artifact. In order to maintain the integrity of business rules for constructed process views, a well-defined set of rules for the view construction is required. Technically, this brings in the challenge to develop a mechanism to generate a process view that correctly preserves the consistency between itself and its underlying business processes.

3 Artifact-Centric Process Model for Business Processes and Views

The concept of modelling artifact-centric processes has been established under the framework proposed in [10] and its formal model can be found in [11]. We extend their artifact-centric business process model to *process views*. The artifact-centric process (ACP) model consists of three core constructs: *artifacts*, *services*, and *business rules*. An *artifact* is a business entity or an object involved in business process(es). Each artifact contains a set of attributes and states. For each artifact, the evolution of states from the initial to the final states describes its lifecycle. A *service* is a task that requires input data from artifact(s) or users, and produces an output by performing an update on artifact(s). We use a *business rule* to associate service(s) with artifact(s) in a *Condition-Action* style. Each rule describes which service is invoked and which state of artifact is changed under what pre-condition and post-condition. We describe the behavior of each artifact in the ACP model by adopting the state machine, and formally define a view of artifact and a view of ACP which is derived from the ACP model.

3.1 Syntax of Artifact-Centric Process Model

In this section, components and syntaxes of artifact-centric process model for business processes are introduced and defined.

Definition 1: (Artifact class). An *artifact class* abstracts a group of artifacts with their data attribute and states. An artifact class C is a tuple (A, S) where,

- $A = \{a_1, a_2, \dots, a_x\}$, $a_i \in A(1 \leq i \leq x)$ is an attribute of a scalar-typed value (string and real number) or an undefined value
- $S = \{s_1, s_2, \dots, s_y\} \cup \{s^{init}\}$, $s_i \in S(1 \leq i \leq y)$ is a state and s^{init} denotes initial state.

Definition 2: (Artifact schema). An *artifact schema* Z contains a set of artifact classes, i.e., $Z = \{C_1, C_2, \dots, C_n\}$ where $C_i \in Z(1 \leq i \leq n)$ is an *artifact class*.

From our business scenario, we define a primary set of original artifact classes used for the product ordering and shipping processes.

- $Order = (\{orderID, customerID, grandTotal\}, \{open_for_item, ready_for_shipping, in_shipping, shipped, billed, closed\})$
- $Shipment = (\{shipID, customerID, shipDate, shipCost\}, \{open_for_shipment, ready_to_dispatch, in_shipping, completed\})$
- $OrderItem = (\{orderID, productID, shipID, qty, price\}, \{newly_added, on_hold, ready_to_ship, added_to_shipment, in_shipping, shipped\})$
- $Invoice = (\{invoiceID, orderID, invoiceDate, amountPaid\}, \{unpaid, paid\})$

We also define some basic predicates over schema Z :

- $defined(C, a)$ if the value of attribute $a \in C.A$ in artifact of class C is defined
- $instate(C, s)$ if the current (active) state of artifact of class C is s . Initially, $instate(C, init)$ implies $\forall x \in C.A, \neg defined(C, x)$. If $instate(C, s)$ then every state in $ancestor(s)$ is also active.

Definition 3: (Business Rule). A business rule regulates which service can be invoked under what pre-condition. The conditional effect is also defined to restrict the post-condition after performing such service. Business rule r can be defined as tuple (λ, β, v) where,

- λ and β are a *pre-condition* and *post-condition*, respectively. Both are defined by a quantifier-free first-order logic formula. For the simplification, it allows only AND logical connective (\wedge) and variables. The formula contains two types of proposition over schema Z : (1) *state proposition* (the *instate* predicate) and (2) *attribute proposition* (the *defined* and scalar comparison operators), e.g., $defined(Order, orderID) \wedge instate(Order, in_shipping) \wedge Order.grandTotal > 10$.
- $v \in V$ is a service to be performed. A service may involve with several artifacts of classes C_1, C_2, \dots, C_y , where $C_i \in Z(1 \leq i \leq y)$.

In order to maintain the existence of valid state changes of an artifact in business rule r , we require that there exists a couple of *instate* predicates of that artifact in both pre-condition and post-condition of r , i.e., we have states $s_x, s_y \in C.S$ such that $instate(C, s_x)$ exists in $r.\lambda$ and $instate(C, s_y)$ exists in $r.\beta$. The state change refers to either a transition from one state to another state, or to itself. Due to page limitation, we only list some business rules in Table 1 which are used for the ordering process in our business scenario.

Table 1. Example of business rules

r1 : Customer c requests to make an Order o	
Pre-condition	$instate(o, init) \wedge \neg defined(o.orderID) \wedge \neg defined(o.customerID) \wedge defined(c.customerID)$
Service	$createOrder(c, o)$
Post-condition	$instate(o, open_for_item) \wedge defined(o.orderID) \wedge defined(o.customerID) \wedge defined(c.customerID) \wedge o.customerID = c.customerID$
r2 : Add OrderItem oi of Product p with a quantity qty to Order o	
Pre-condition	$instate(o, open_for_item) \wedge instate(oi, init) \wedge defined(p.productID) \wedge defined(oi.productID) \wedge \neg defined(oi.orderID) \wedge defined(oi.qty) \wedge \neg defined(oi.price)$
Service	$addItem(o, p, oi)$
Post-condition	$instate(o, open_for_item) \wedge instate(oi, newly_added) \wedge defined(oi.orderID) \wedge defined(oi.price)$
r3 : Complete Order o	
Pre-condition	$instate(o, open_for_item) \wedge o.grandTotal > 0$
Service	$completeOrder(o)$
Post-condition	$instate(o, ready_for_shipping)$
r4 : Ship Shipment s which contains OrderItem oi of Order o	
Pre-condition	$instate(o, ready_for_shipping) \wedge instate(oi, added_to_shipment) \wedge instate(s, ready_to_dispatch) \wedge defined(oi.shipID) \wedge defined(s.shipID) \wedge oi.shipID = s.shipID$
Service	$startShipping(s, o, oi)$
Post-condition	$instate(s, in_shipping) \wedge instate(oi, in_shipping) \wedge instate(o, in_shipping)$
r5 : Clear Invoice v for Order o	
Pre-condition	$instate(o, billed) \wedge instate(v, unpaid) \wedge defined(v.orderID) \wedge o.orderID = v.orderID \wedge o.grandTotal = v.amountPaid$
Service	$payInvoice(v, o)$
Post-condition	$instate(o, closed) \wedge instate(v, paid)$

As we can see that pre- and post- conditions of each business rule contain two groups of conditioning: *attribute* (attribute's value evaluation) and *state*. We also classify state conditioning into two types by determining the existence of *instate* predicate in the pre- and post- conditions of a business rule. The first type is classified by the case that a change of state occurs for only one artifact, e.g., rules $r1$ and $r3$. The second type refers to changes of states of multiple artifacts, i.e., more than one pair of *instate* predicates appears in pre- and post-conditions of a single business rule for different artifacts. This type indicates that a single business rule is used to induce multiple state changes such that each change occurs once on each artifact, e.g. rule $r5$ changes for the *Order* artifact from the *billed* state to the *closed* state, and simultaneously changes from the *unpaid* state to the *paid* state for the *Invoice* artifact.

Definition 4: (Artifact-Centric Process Model or ACP model). Let Π denote an artifact-centric process model, and it is tuple (Z, V, R) where Z is an artifact schema, V and R are sets of services and business rules over Z , respectively.

In this paper, we assume that the ACP model is correct and valid, so we do not consider the verification of the model. Actually, the verification can be adopted from the work presented in [11]. Our focus is on the model of views and their construction.

3.2 View Definitions

As discussed earlier, process views for artifact-centric business processes are defined and constructed according to view requirements for particular roles of the organizations. The view requirement of a particular view specifies a hierarchal structure of states of an artifact and their visibility. As such, we introduce composite states to define and represent a view of an artifact.

Definition 5: (Role). A role defines the capability of a group of users. Different roles may have different views/accesses to artifacts. The role can be used to differentiate an external organization from a home organization, inter-organizational units, or different groups within an organizational unit. We denote role domain $L = \{l_1, l_2, \dots, l_x\}$, where $l_i \in L (1 \leq i \leq x)$ is a role of users who involve in the business processes.

Definition 6: (Artifact view). Given artifact class C , we denote C^l for a *view* of C for role $l \in L$, and it is tuple (A^l, S^l, pc) where $A^l = C.A$, S^l is a set of states that are defined as nodes in a hierarchal tree structure (state tree), and $pc \subseteq S^l \times S^l$ is a finite set of parent-child relations.

We define predicate $child(s_x, s_y)$ to mean that state s_y is a child of state s_x , and function $children: S \rightarrow 2^S$ to define for each state its set of child states. In addition, we define function $descendant$ as the reflexive transitive closure of $children$. We can say that state s_y is an *ancestor* of state s_x in the state tree if $s_x \in descendant(s_y)$. Correspondingly, let $ancestor: S \rightarrow 2^S$ be the function that defines for each state its set of ancestors, i.e., $ancestor(s_x)$ returns a set of all ancestors of s_x .

As we are required to ensure the hierarchal structure of the state tree, we add the root state to a set of states for each artifact class. The root state is ancestor of every state in S . For simplicity, we do not show the root state in a class.

Note that for *artifact view* C^l , we write $s_y: \{s_1, s_2, \dots, s_x\}$ where s_y and $s_i \in C^l.S (1 \leq i \leq x)$ to denote composite state s_y together with its nested states $\{s_1, s_2, \dots, s_x\}$, e.g., a set of states of the *Order* artifact for the *Sale* role in Figure 2 (c) can be defined as $\{created: \{init, open_for_item\}, ready_for_shipping, in_processing: \{delivering: \{in_shipping, shipped\}, billed\}, closed\}$.

Definition 7: (ACP view). Given role $l \in L$ and ACP model $\Pi = (Z, V, R)$, we denote Π^l for the ACP view of Π for role l , and it is tuple (Z^l, V^l, R^l) , where Z^l , V^l , and R^l is a set of *views* of artifact classes for role l , the services, and business rules over Z^l , respectively, such that for every view $C^l \in Z^l$ of artifact class C then $C \in Z$.

An ACP view is an abstract process model derived from its corresponding business ACP model which represents the original (base) process, namely *operational view*, or an existing view. We also use the term *operational view* of an artifact for the most detailed view of the artifact. In order to generate ACP views, we propose the so called *state condensation* technique that constructs a view of an artifact by abstraction of states of the artifact, and it is described in Section 4. One artifact may have different

views depending on different view requirements specified by different roles. In short, an ACP view for a particular role will have for each artifact its view for that role.

Without loss of generality, we assume the uniqueness of artifact’s state in a schema, i.e., a state belongs to only one artifact class. Given ACP model $\Pi = (Z, V, R)$, we can simply use the abbreviated notation $\Pi.s_x$ instead of using $\Pi.Z.C.s_x$ as well as the state set $\Pi.S$ is denoted for $\bigcup_i^{|Z|} \Pi.Z.C_i.S$. It also has the implication that when referring a state in Π , it will match with only one corresponding artifact class in $\Pi.Z$. This abbreviated uses of notation are also applied for ACP views.

3.3 Artifact Lifecycle Model

Each artifact has its own life-cycle showing how its states are changed through the business processes. We adopt the state machine to capture a lifecycle of each individual artifact, namely *Artifact lifecycle model (ALM)*. Each artifact has its corresponding ALM. Given an ACP model, a lifecycle model of an artifact can be generated by deriving from corresponding business rules that are used to induce state transitions of the artifact.

Definition 8: (Artifact Lifecycle Model). An *Artifact Lifecycle Model* defines the state transition of an artifact class. Given ACP model $\Pi = (Z, V, R)$ and artifact class $C_i = (A_i, S_i)$ where $C_i \in Z$, an artifact lifecycle model for C_i , denoted as LM_{C_i} , can be defined as tuple (C_i, T) , where

- $T \subseteq C_i.S \times R \times C_i.S$ is a 3-ary transition relation where R is a set of business rules. A transition $t = (s_s, r_i, s_t) \in T$ means that the state of the artifact will change from source state s_s to target state s_t where $s_s, s_t \in C_i.S$, if the pre-condition λ of business rule r_i holds.
- T^* is reflexive transitive closure of T . We write $s_i T^* s_j$ if there exists sequence of transitions from s_i to s_j , i.e., state s_j can be reached from state s_i by some business rules in R .

Figure 3 depicts the artifact lifecycle diagram for each artifact class of our business scenario. A label on a transition (arrow) denotes a business rule that corresponds to a transition, and it is one-to-one correspondence. We can see that rule r_4 induces three transitions of *Order*, *Shipment*, and *OrderItem* artifact lifecycles.

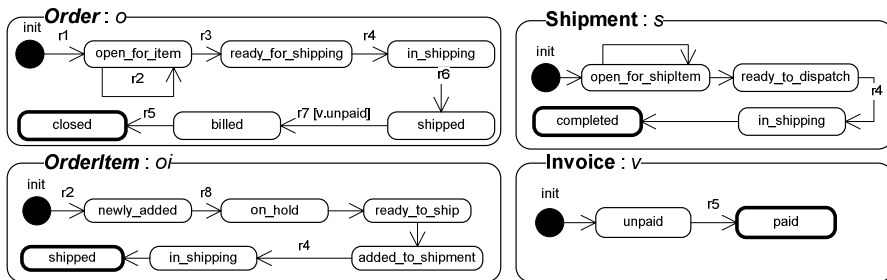


Fig. 3. Artifact Lifecycle Diagrams

Note that ALM is also used for describing the lifecycle of an artifact view in ACP views. We define an artifact lifecycle model for *view* C_i^l of artifact class C_i , denoted as $LM_{C_i^l}$, and it is tuple (C_i^l, T) .

Definition 9: (Ordering relation). Given artifact lifecycle model $LM_{C_i} = (C_i, T)$ for artifact class (or *view*) C_i , state $s_x \in C_i.S$ is *before* state $s_y \in C_i.S$ if there is a sequence from state s_x to state s_y , i.e., $s_x T^* s_y$. We denote $s_x < s_y$ for their ordering relation. If $s_x T^* s_y$ and $s_y T^* s_x$, i.e., $s_x < s_y$ and $s_y < s_x$, then every state in sequences from s_x to s_y and s_y to s_x is in a loop. If $\neg(s_x < s_y) \wedge \neg(s_y < s_x)$, then state s_x and s_y are independent and we denote it as $s_x \parallel s_y$ for their ordering relation.

4 View Construction Framework for Artifact-Centric Business Processes

In this section, we propose a framework for constructing process views for artifact-centric business processes. The framework consists of operational business process model and its constructed views. We formally introduce definitions, syntaxes, rules, and related functions that will be used in the framework, and present a *condensation technique* to construct views based on its underlying process models with role-based view configurations. The details of this technique are described along with related cases regarding our motivating example. In addition, *process view consistency rules* are also defined for preserving the consistency between a constructed process view and its underlying process.

The condensation process is divided into two steps: (1) *State composition* and (2) *State hiding*. The *state composition* step is straightforward by nesting specified states in a composite state. The *state hiding* hides a specified set of states. The process of hiding states is more complex than composing states as business rules that has any reference to the hidden states will be affected. We also define *view consistency rules* to preserve the structural and behavioral consistencies between constructed view and its underlying view when applying those two steps.

The relation between a constructed view and its underlying view is defined by a *view transformation* function which provides the mapping between them. Here, we assume the existence of ACP view set $\Sigma = \{\Pi, \Pi^{l_1}, \Pi^{l_2}, \dots, \Pi^{l_x}\}$, where Π is the *operational view* of ACP model and Π^{l_i} is an ACP View for role $l_i \in L(1 \leq i \leq x)$, and Σ forms a hierarchal structure having Π as its root, such that for any two ACP views in Σ , the state ordering of them must be consistent.

Consistency Rule 1: (State ordering relation preservation). Let Π^{l_1} and Π^{l_2} be ACP view for role l_1 and role l_2 , respectively. For any two states that belong to two views of the same artifact class C_i in both Π^{l_1} and Π^{l_2} , the ordering relation between them must be consistent, i.e.,

If $s_x, s_y \in \Pi^{l_1}.S \cap \Pi^{l_2}.S$ such that $s_x < s_y$ in Π^{l_1} , then $s_x < s_y$ in Π^{l_2} , or if $s_x, s_y \in \Pi^{l_1}.S \cap \Pi^{l_2}.S$ such that $s_x \parallel s_y$ in Π^{l_1} , then $s_x \parallel s_y$ in Π^{l_2} .

This consistency rule ensures that the transitions corresponding to business rules of two ACP views are consistent.

Definition 10: (View transformation). Given ACP view set Σ for ACP model $\Pi = (Z, V, R)$, the *view transformation* $vt = sh \circ sc: \Sigma \times SR^+ \times SR^- \rightarrow \Sigma$ is a composite function where state composition function sc and state hiding function sh are composed, i.e., the result ACP view after applying state composition function is then applied with the state hiding function. Function $vt(\Pi, sr_l^+, sr_l^-)$ returns a role-based view, i.e., Π^l , of ACP model Π that constructed based on *state composition requirement* sr_l^+ and *state hiding requirement* sr_l^- for role l .

We use the term *view configuration* for the combined set of state composition and state hiding requirements. To limit the scope of the paper, we also assume that the view configuration that is used to transformed from ACP view Π^i to ACP view Π^j contains non-conflict set of state composition and hiding requirements, i.e., the view shall be constructed by satisfying every requirement and preserving every view consistency rule.

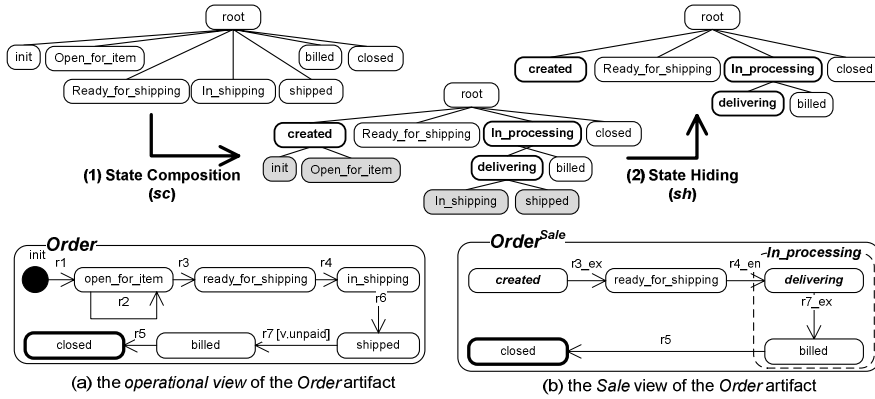


Fig. 4. Two views of the Order artifact

Figure 4 illustrates an example of two different views of the *Order* artifact regarding our motivating example. Figure 4 (b) shows an abstraction of the *Order* artifact for the *Sale* role that is transformed from its operational view in Figure 4 (a) based on two specified view configuration requirements: (1) *composition requirement* for nesting the *in_shipping* and *shipped* states under the *delivering* composite state and then nesting the *in_shipping* and *billed* states under the *in_processing* composite state, (2) *hiding requirement* for hiding the *in_shipping* and *shipped* states under the *delivering* state. Here, we also name an *intermediate ACP view* for a view of an artifact that is generated by state composition function (sc), and will be further used as an input of state hiding function (sh).

In order to preserve the structural and behavioral consistencies of the operational ACP, the condensation process may incur a rearrangement of state transition of an artifact. The rearrangement of a transition is achieved by a modification of corresponding

business rule that induces such transition. The detail of modification is described in Section 4.3.

4.1 State Composition

State composition refers to the process of nesting states at same level under the new defined *composite state*. The composition of states will amend the structure of the state tree by inserting the new composite state into the hierarchy. Generally, defining a new composite state and inserting it into the tree hierarchy can be done iteratively in bottom-up manner.

Definition 11: (State composition). Given ACP view set Σ for ACP model $\Pi = (Z, V, R)$, the state composition $sc: \Sigma \times SR^+ \rightarrow \Sigma$ is a bijective function that maps one ACP view onto another ACP view in which the state trees of views of artifact classes are restructured and corresponding business rules are rewritten according to the state composition requirement set $SR^+ = \{sr_n^+\}$, and a single *state composition requirement* $sr_n^+ = \{\oplus_{C_1}, \oplus_{C_2}, \dots, \oplus_{C_x}\}$, where $\oplus_{C_i}(1 \leq i \leq x) = \{(s_{j_1}, S_{k_1}), (s_{j_2}, S_{k_2}), \dots, (s_{j_x}, S_{k_x})\}$ is a set of *atomic composition requirements* for artifact $C_i \in \Pi.Z(1 \leq i \leq x)$. The atomic requirement $(s_j, S_k) \in \oplus_{C_i}(1 \leq j \leq y, 1 \leq k \leq z)$ defines the state composition s_j for all states in S_k ($|S_k| > 1$) of artifact class C_i . Composite state s_j is inserted into the state tree as a parent of all nested states in S_k in the constructed ACP view.

An *atomic composition requirement* (s_j, S_k) is said to be *valid* if every state to be composed in S_k , belonging to the same artifact class, is at the same height of the state tree and owns the same parent. For example, we assume that ACP model Π contains the original *Order* artifact in Figure 4 (a) and let a single composition requirement sr^+ be $\{\oplus_{Order}\}$, where $\oplus_{Order} = \{(delivering, \{in_shipping, shipped\}), (in_processing, \{delivering, billed\}), (created, \{init, open_for_item\})\}$. Then, we will have an intermediate ACP view $\Pi^{Sale'}$ for role *Sale* after applying the state composition function sc to Π with requirement sr^+ .

4.2 State Hiding

The process of hiding states is an important step of the condensation process. Unlike the state composition which mainly focuses on the state tree structure of artifact, hiding states deals with behavior of the artifact. The hidden state must be invisible in (or removed from) the tree structure and the business rule that such state is referenced.

Definition 12: (State hiding). Given ACP view set Σ for ACP model $\Pi = (Z, V, R)$, the state hiding $sh: \Sigma \times SR^- \rightarrow \Sigma$ is a bijective function that maps one ACP view onto another ACP view in which the state tree and transitions of views of artifact classes are restructured and corresponding business rules are modified according to the state hiding requirement set $SR^- = \{sr_n^-\}$, and a single *state hiding requirement* $sr_n^- = \{\ominus_{C_1}, \ominus_{C_2}, \dots, \ominus_{C_x}\}$, where $\ominus_{C_i}(1 \leq i \leq x) \subseteq C_i.S$ is a set of states to be hidden for artifact $C_i \in \Pi.Z(1 \leq i \leq x)$.

The set of states \ominus_{C_i} to be hidden is said to be *valid* if every state in \ominus_{C_i} has its own parent and the parent is not the *root* of the state tree. This restriction is used to ensure that every state to be hidden is under some composite state, which is not the

root, in the tree. For example in Figure 4, after the state composition step, states *init*, *open_for_item*, *in_shipping*, and *shipped* of the *Order* artifact can be hidden in the *Sale*'s view. This implies that if we want to hide such states, we must first compose them and then we can hide them, e.g., the *delivering* composite state is created for the purpose of allowing the *in_shipping* and the *shipped* states to be hidden. If any composite state is to be hidden, all of its descendant states must be also hidden to ensure that a constructed view preserves the hierarchical structure.

Consistency Rule 2: (Hierarchy preservation). Let Π^{l_1} be ACP view for role l_1 and Π^{l_2} be ACP view for role l_2 that is constructed based on Π^{l_1} . For any state s_x that belongs to the same artifact class C_i in both Π^{l_1} and Π^{l_2} , the set of ancestors S_1 of s_x in Π^{l_1} is a subset of the set of ancestors S_2 of s_x in Π^{l_2} , and the states in S_1 but not in S_2 do not exist in Π^{l_1} , i.e.,

If $s_x \in \Pi^{l_1}.S \cap \Pi^{l_2}.S$ then $ancestor(\Pi^{l_1}.s_x) \subseteq ancestor(\Pi^{l_2}.s_x)$ and $tor(\Pi^{l_2}.s_x) \setminus ancestor(\Pi^{l_1}.s_x) \rightarrow s_y \notin \Pi^{l_1}.S$.

We take ACP view for role *Sale* from Figure 4 (b) as an example, and choose the *billed* state to be validated for the hierarchy consistency between the original state tree and the state tree after state composition for role *Sale*. The set of *ancestors* of the *billed* state for the view in Figure 4 (b) is $\{in_processing, root\}$, while for the view in Figure 4 (a) is $\{root\}$. As $\{root\} \subseteq \{in_processing, root\}$, and the *in_processing* state does not appear in the original state tree, we say that the *billed* state preserves the hierarchy consistency between the *Sale*'s view and its underlying original view. If every state that exists in both ACP views preserves the hierarchy consistency, then the state tree of a constructed view is consistent with its base view. This rule is also used to ensure that if any composite state is created, e.g., states *created*, *delivering*, and *in_processing*, then it must be correctly structured in the state tree.

4.3 Modification of Business Rules

We classify the modification of business rules into two categories regarding two types of conditionings which can be defined in business rules: (1) *state conditioning modification* and (2) *attribute conditioning modification*. The *state conditioning modification* is the process of hiding specified states and rearranging all effected transitions by considering *instate* predicates in pre- and post-conditions of the business rule that is related to the hidden state. The *attribute conditioning modification* updates every attribute condition of the business rule that is affected by hidden states. A business rule may be removed if it is no longer used in the constructed view.

Definition 13: (Modified business rule). Let $\Pi^{l_{y'}} = sc(\Pi^{l_x}, s \tau_{l_y}^+)$ be an intermediate ACP view for role l_y constructed based on ACP view Π^{l_x} with state composition requirement $sr_{l_y}^+$, and $sh(\Pi^{l_{y'}}, sr_y^-)$ be the function that returns ACP view $\Pi^{l_y} = (Z^{u_y}, V^{l_y}, R^{l_y})$ for role l_y that is constructed based on $\Pi^{l_{y'}}$ with state hiding requirement $sr_{l_y}^-$, we have a *modified business rule set*, denoted as $R_{mod}^{l_y}$, such that

each rule $r \in R_{mod}^{ly}$ is the modified version of its original business rule in R^{ly} according to the effect of state hiding requirement sr_{ly}^- .

The modification of business rules must preserve the behavioral consistency between the constructed process view and its underlying base view. We define and describe related consistency rules to be used within the modification processes.

4.3.1 State Conditioning Modification

It is the fact that hiding any state of an artifact will break up the transition relation between such hidden state and other state, i.e., some paths or ordering relations in a lifecycle of the artifact are broken. To preserve such relations for maintaining the consistent behavior of the artifact lifecycles between ACP views, we consider the rearrangement of transitions that are affected by a hidden state, i.e., modifying state conditions of business rules that correspond to those transitions.

In order to explain the modification processes, we combine state tree with the lifecycle diagram of an artifact. Figure 5 (a) illustrates an example of combined diagram of the *Order* artifact. The dotted arrow indicates the hidden state transition and its corresponding hidden business rule, while the normal arrow indicates a rearranged transition with its modified business rule.

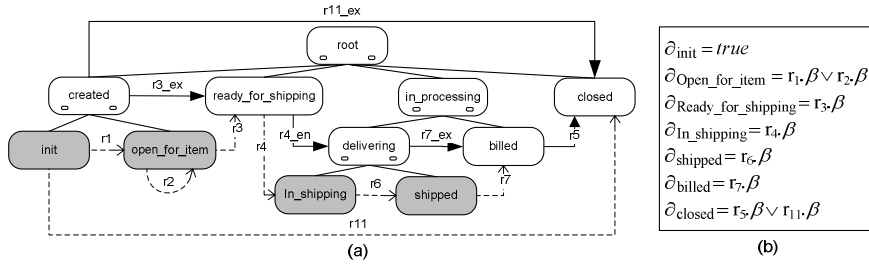


Fig. 5. (a) combined diagram for the *Order* artifact (b) compensating conditions

The modification process starts with finding *entry* and *exit* transitions of every hidden state and their parent composite state. If any transition is connected from/to a hidden state that is nested under the composite state to/from a non-hidden state, then that transition is rearranged to replace the hidden state with the composite state. If the transition is used to connect between nested hidden states, then that transition is hidden. We define two auxiliary functions for finding the entry and exit transitions of the composite state. Function $entry(s_c)$ returns a set of entry transitions T_{en} of composite state s_c , such that every transition in T_{en} has the source state which is not a descendant of s_c and the target state which is a descendant of s_c . Function $it(s_c)$ returns a set of exit transitions T_{ex} of composite state s_c , such that every transition in T_{ex} has the source state which is a descendant of s_c and the target state which is not a descendant of s_c . For example in Figure 5 (a), $entry(delivering)$ returns $\{\{ready_for_shipping, r_4, in_shipping\}\}$ and $exit(delivering)$ returns $\{\{shipped, r_7, billed\}\}$.

Once we obtain entry and exit transitions from the *entry* and *exit* functions, respectively, then we rearrange them to the composite state by modifying business rules that corresponds to them, e.g., r_4 is modified to r_{4-en} and r_7 is modified to r_{7-ex} . The state propositions in their pre- and post- conditions are changed accordingly, e.g., for rule r_{4-en} , the *in_shipping* state is substituted by the *delivering* state for *instate* predicate in the post-condition of r_4 , and for rule r_{7-ex} , the *shipped* state is substituted by the *delivering* state for *instate* predicate in the pre-condition of r_7 . However, transitions for business rules r_1 and r_2 are hidden as they do not exist in the result set of either *entry* or *exit* function. We can say that those hidden states and hidden transitions are encapsulated in the *created* composite state. Note that hiding a business rule may have a propagate modification effect to other artifact if such rule is used to induce any transition of the other artifact, e.g., r_2 also induces the transition from its *init* state to *newly_added* state of the *OrderItem* artifact, so such transition must be hidden as well as those two states.

Due to the limitation of our scope discussed earlier, we consider only non-conflict set of state composition and hiding requirements, i.e., the view configuration must be self-completed, e.g., those two *init* and *newly_added* states must be predefined in the requirement in order to be hidden. Here, we also define another two consistency rules that the modification process must conform to.

Consistency Rule 3: (Atomicity of composite state preservation). Let $\Pi^{l_2'} = sc(\Pi^{l_1}, sr_{l_2}^+)$ be an intermediate ACP view for role l_2 constructed based on ACP view Π^{l_1} with state composition requirement $sr_{l_2}^+$, and let $\Pi^{l_2} = sh(\Pi^{l_2'}, sr_{l_2}^-)$ be an ACP view for role l_2 that is constructed based on $\Pi^{l_2'}$ with state hiding requirement $sr_{l_2}^-$. For any composite state that belongs to the view of artifact class C_i in $\Pi^{l_2'}$, and every of its descendant that is hidden in Π^{l_2} , the *entry transitions* and *exit transitions* to/from the composite state must be consistent if such composite state exists in both Π^{l_2} and $\Pi^{l_2'}$, i.e.,

Given artifact lifecycle models $LM_{C_i^{l_2'}} = (C_i^{l_2'}, T)$ for a view of artifact class C_i in $\Pi^{l_2'}$, and $LM_{C_i^{l_2}} = (C_i^{l_2}, T)$ for a view of artifact class C_i in Π^{l_2} , if $s_x \in \Pi^{l_2'}.S \cap \Pi^{l_2}.S$ such that $|children(\Pi^{l_2'}.s_x)| > 1 \wedge |children(\Pi^{l_2}.s_x)| = 0$, then

- (1) $\exists s_m \in \Pi^{l_2'}.S \cap \Pi^{l_2}.S, \exists s_n \in descendant(\Pi^{l_2'}.s_x), \exists r_p \in \Pi^{l_2'}.R, \exists r_q \in \Pi^{l_2}.R,$
 $(s_m, r_p, s_n) \in LM_{C_i^{l_2'}}.T, (s_m, r_q, \Pi^{l_2}.s_x) \in LM_{C_i^{l_2}}.T, s_n \notin \Pi^{l_2}.S,$ and
- (2) $\exists s_m \in descendant(\Pi^{l_2'}.s_x), \exists s_n \in \Pi^{l_2'}.S \cap \Pi^{l_2}.S, \exists r_p \in \Pi^{l_2'}.R, \exists r_q \in \Pi^{l_2}.R,$
 $(s_m, r_p, s_n) \in LM_{C_i^{l_2'}}.T, (\Pi^{l_2}.s_x, r_q, s_n) \in LM_{C_i^{l_2}}.T, s_m \notin \Pi^{l_2}.S,$ and
- (3) $\forall s_n (s_n \in descendant(\Pi^{l_2'}.s_x) \rightarrow s_n \notin \Pi^{l_2}.s_x)$

Condition (1) preserves the consistency of every *entry transition* that is changed from other non-hidden state to any nested hidden states under the composite state. Correspondingly, Condition (2) preserves *exit transitions*. Condition (3) ensures that every nested hidden state is removed and it also implies that every transition between those states is hidden. This ensures that the atomicity of the composite state is preserved in the constructed ACP view as well as the integrity of business rules of the ACP view is maintained.

Consistency Rule 4: (Business rule – transitions of multiple artifacts preservation). Let Π^{l_1} be ACP view for role l_1 and Π^{l_2} be ACP view for role l_2 that is constructed based on Π^{l_1} . If a business rule induces transitions of multiple artifacts and any of these transitions in one artifact is hidden in Π^{l_2} then such rule and its induced transitions in the other artifacts must be hidden in Π^{l_2} , i.e.,

Given artifact lifecycle models $LM_{C_i l_1} = (C_i^{l_1}, T)$ for a view of one artifact class C_i in Π^{l_1} , and $LM_{C_i l_2} = (C_i^{l_2}, T)$ for a view of artifact class C_i in Π^{l_2} , if $\exists r \in \Pi^{l_1}.R$, $\exists s_x, s_y \in \Pi^{l_1}.S$, $(s_x, r, s_y) \in LM_{C_i l_1}.T$, $s_x, s_y \notin \Pi^{l_2}.S$ then $\forall C_m \in \Pi^{l_1}.Z \cap \Pi^{l_2}.Z$, $\forall s_j, s_k \in \Pi^{l_1}.S$, $(s_j, r, s_k) \in LM_{C_m l_1}.T \rightarrow s_j, s_k \notin \Pi^{l_2}.S \wedge r \notin \Pi^{l_2}.R$

Revisiting the lifecycles of the *Order* and *OrderItem* artifacts in Figure 3, if we hide the *open_for_item* state of the *Order* artifact, then business rule r_2 is hidden, and consequently, the *init* state, the *newly_added* state, and their transition of the *OrderItem* artifact must also be hidden.

4.3.2 Attribute Conditioning Modification

Before we explain the detail of the *attribute conditioning modification*, we would like to explain the reason why this kind of modification is needed. As already discussed, when any state is hidden and *entry/exit* transition is rearranged to its composite state, the source state of the rearranged transition is changed from the concrete state to the composite state. In order to compromise the loss of the details about specific states when composing and hiding states, we attempt to maintain the condition of each business rule that corresponds to the rearranged transition as most specific as possible. As such, we propose to *strengthen* the attribute condition of the business rule.

Revisiting Figure 5, when the *init* and *open_for_item* states are hidden, business rule r_3 and r_{11} are modified to r_{3-ex} and r_{11-ex} , respectively. Both r_{3-ex} and r_{11-ex} will have the more abstract state condition in their pre-conditions, i.e., *in-state(created)* instead of *instate(open_for_item)* in r_3 and *instate(init)* in r_{11} . So, we will need to strengthen their attribute condition in both r_{3-ex} and r_{11-ex} . Informally, strengthening an attribute condition of business rule for a particular transition is done by extracting other attribute condition in a post-condition of business rule that is used to move artifact into the source state of that transition.

Referring to the artifact lifecycle, we can see that when the artifact is in a particular state, it holds a post-condition of one of the transitions that bring it into that state, e.g., in Figure 5, if the *Order* artifact is in the *open_for_item* state, there will be an implication that either post-condition of the business rules r_1 or r_2 holds as they induce the transition into the *open_for_item* state.

Here, we define *compensating condition* for a group of attribute propositions of such post-condition that holds for each state.

Definition 14: (Compensating condition). Given ACP $\Pi = (Z, V, R)$ and artifact lifecycle model $LM_{C_i} = (C_i, T)$ for artifact $C_i \in Z$, a *compensating condition* on state $s_j \in C_i.S$, denoted as ∂_{s_j} , is the logical disjunction of every attribute proposition of C_i in post-condition β of every business rule $r \in R$ that triggers a transition from any state in $C_i.S$ to state s_j .

Figure 5 (b) lists the set of compensating conditions for every state of the *Order* artifact. These conditions are extracted from the business rules defined in Table 1, e.g., for the *open_for_item* state, we have compensating condition $\partial_{open_for_item} = (r_1.\beta \vee r_2.\beta) = ((defined(o.orderID) \wedge defined(o.customerID) \wedge equal(o.customerID, c.customerID)) \vee false)$. As the post-condition of business rule r_2 contains no attribute propositions of the *Order* artifact, so Boolean *false* is substituted for $r_2.\beta$. As discussed, at least one of the attribute post-condition of business rule r_1 or business rule r_2 must hold when the artifact is in the *open_for_item* state.

In order to strengthen the pre-condition of such rule, we add the compensating condition to its pre-condition. Now, we define a *modified condition* on transition $t_k \in T$, denoted as $\partial_{s_j}^{t_k}$, with state s_j as its source state, such that $\partial_{s_j}^{t_k}$ holds compensating condition ∂_{s_j} and pre-condition λ of corresponding business rule r_y for t_k , i.e., $\partial_{s_j}^{t_k}$ holds $(r_y.\lambda \wedge \partial_{s_j})$. Thus, we can see that the pre-condition of modified business rule for transition t_k is stronger than its original rule.

Revisiting the example in Figure 5, the pre-condition of a modified business rule for each rearranged transition of the *Order* artifact is: $\partial_{created}^{(created, r_{3-ex}, ready_for_shipping)}$ holds $(r_3.\lambda \wedge \partial_{open_for_item})$, $\partial_{delivering}^{(delivering, r_{7-ex}, billed)}$ holds $(r_7.\lambda \wedge \partial_{shipped})$, $\partial_{delivering}^{(created, r_{11-ex}, closed)}$ holds $(r_{11}.\lambda \wedge \partial_{init})$, while $\partial_{ready_for_shipping}^{(ready_for_shipping, r_{4-en}, delivering)}$ holds the original pre-condition of r_4 as the source state of its transition is not changed. By adding compensating conditions, at least two transitions in the *Order* artifact for business rules r_{11-ex} and r_{3-ex} are different even they both have the same source state.

Note that if a business rule induces multiple state transitions of different artifacts, then every compensating condition on each transition in each artifact is added to the pre-condition of the modified business rule. This modification process conforms to consistency *Rule 5*.

Consistency Rule 5: (Attribute condition preservation). Let $\Pi^{l_2'} = sc(\Pi^{l_1}, sr_{l_2}^+)$ be an intermediate ACP view for role l_2 constructed based on ACP view Π^{l_1} with state composition requirement $sr_{l_2}^+$, and let $\Pi^{l_2} = sh(\Pi^{l_2'}, sr_{l_2}^-)$ be an ACP view for role l_2 that is constructed based on $\Pi^{l_2'}$ with state hiding requirement $sr_{l_2}^-$. For any rearranged *entry* or *exit* transition of a composite state in Π^{l_2} , the attribute condition of the pre-condition of a business rule for such transition must hold a *modified condition* on that transition: (1) an attribute condition of the pre-condition of its original business rule in $\Pi^{l_2'}$ and (2) a compensating condition on the source state of that transition. However, the post-conditions of them must be identical, i.e.,

- (1) $\exists s_i \in \Pi^{l_2'}.S \cap \Pi^{l_2}.S, \exists s_j \in \Pi^{l_2'}.S, \exists s_m \in ancestor(s_j), \exists r_p \in \Pi^{l_2'}.R, \exists r_q \in \Pi^{l_2}.R, (s_i, r_p, s_j) \in LM^{l_2'}_{c_i}.T, (s_i, r_q, s_m) \in LM^{l_2}_{c_i}.T, s_j \notin \Pi^{l_2}.S \rightarrow (attr(r_q.\lambda) = attr(r_p.\lambda)) \wedge (attr(r_q.\beta) = attr(r_p.\beta))$, or
- (2) $\exists s_i \in \Pi^{l_2'}.S \cap \Pi^{l_2}.S, \exists s_j \in \Pi^{l_2'}.S, \exists s_m \in ancestor(s_j), \exists r_p \in \Pi^{l_2'}.R, \exists r_q \in \Pi^{l_2}.R, (s_j, r_p, s_i) \in LM^{l_2'}_{c_i}.T, (s_m, r_q, s_i) \in LM^{l_2}_{c_i}.T, s_j \notin \Pi^{l_2}.S \rightarrow (attr(r_q.\lambda) = attr(r_p.\lambda \wedge \partial_{s_j})) \wedge (attr(r_q.\beta) = attr(r_p.\beta))$, where $attr(r.y)$ is the function that

returns the attribute propositions in a conditional statement $y \in \{\lambda, \beta\}$ of business rule r , and λ, β is the pre-condition and post-condition of r , respectively.

Condition (1) is used to preserve the consistency of an *entry transition* in which a set of attribute conditions in the pre-conditions of them must be identical, as well as for the post-condition. Condition (2) is for an *exit transition* where a set of attribute conditions in the post-conditions of them must be identical, but stronger condition is allowed for the pre-conditions by adding the *compensating condition* on the source state of the transition.

5 Related Work and Discussion

The concept of business artifacts was firstly introduced in [7] with the modelling concept of artifact lifecycles. Gerede and Su [9] presented a formal model for artifact-centric business process specifications. Artifact classes are represented by adding object oriented classes with states, and guarded finite state automata is used to capture the logic of entities that carry out the work in a business model. Similarly, [11] focuses on the evolution of the business process logic by using services to model activities and a set of business rules to declaratively capture and represent a business model. Their work is in accordance with the four-dimensional framework laid in [10] for the specification of processes where business artifacts, artifact lifecycles, services, and associations between services and artifacts are constructed for describing business processes. The artifact-centric process model presented in our work is adapted based on their work with an additional specification to capture nesting states for constructing views of an artifact.

In the area of business process views, Zhao, Liu, Sadiq, and Kowalkiewicz [3] proposed the process view approach based on the perspective of role-based perception control. A set of rules on consistency and validity between the constructed process views and their underlying process view is defined. Many other closely-related works [2, 3, 5, 6] also presented approaches for defining and constructing process views, i.e., abstracted process, based on the consistency rules and constraints to support security/privacy requirements and to facilitate intra/inter organizational collaboration of business processes. Compared with our work, their work aims at process view construction for traditional activity-centric business processes while we propose the view framework for artifact-centric business processes.

Küster, Ryndina, and Gall [12] proposed a technique for generating a compliant business process model from a set of given reference object life cycles. Compared with our work, they transform the object-oriented process into the activity-centric process, while our work generates artifact-centric process views. Hull, Narendra, and Nigam [1] proposed a new approach to interoperation of organizations hubs based on business artifacts. It provides a centralized, computerized rendezvous point, where stakeholders can access data of common interest and check the current status of an aggregate process. They proposed three types of access restriction for stakeholders, namely *window*, *view*, and *CRUD*. *Window* provides a mechanism to restrict which artifacts a stakeholder can see. *View* provides a mechanism to restrict what parts of an artifact a stakeholder can see. *CRUD* is used to restrict the ways that stakeholders can read and modify artifacts. As far as process views are concerned, they did preliminary work on individual artifacts and did not consider business rules. No technique on

view construction of business processes is proposed in their work. In contrast, we take business rules of business processes into account and propose a view condensation technique. We also define a comprehensive set of consistency rules to preserve the structural and behavioural correctness between constructed view and its base model.

6 Conclusion and Future Work

This paper presents a novel view framework for artifact-centric business processes. It consists of a process view model, a set of consistency rules, and the construction approach for building process views. The formal model of artifact-centric business processes and views, namely ACP, is defined and used to describe artifacts, services, business rules that control the processes, as well as views. We develop a mechanism of process view construction to derive views from underlying process models according to view requirements. Consistency rules are also defined to preserve the consistency between constructed view and its underlying process. Our future work will apply the framework for tracking and monitoring artifact instances in the context of process views.

References

1. Hull, R., Narendra, N.C., Nigam, A.: Facilitating Workflow Inter-operation Using Artifact-Centric Hubs. In: Baresi, L., Chi, C.-H., Suzuki, J. (eds.) ICSOC-ServiceWave 2009. LNCS, vol. 5900, pp. 1–18. Springer, Heidelberg (2009)
2. Eshuis, R., Grefen, P.: Constructing customized process views. *Data & Knowledge Engineering* 64, 419–438 (2008)
3. Zhao, X., Liu, C., Sadiq, W., Kowalkiewicz, M.: Process View Derivation and Composition in a Dynamic Collaboration Environment. In: CoopIS 2008, pp. 82–99 (2008)
4. Liu, C., Li, Q., Zhao, X.: Challenges and opportunities in collaborative business process management. *Information System Frontiers* (May 21, 2008)
5. Liu, D., Shen, M.: Workflow modeling for virtual processes: an order-preserving process-view approach. *Information Systems* (28), 505–532 (April 2003)
6. Chebbi, I., Dustdar, S., Tata, S.: The view-based approach to dynamic inter-organizational workflow cooperation. *Data & Knowledge Engineering* 56, 139–173 (2006)
7. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. *IBM Systems Journal* 42(3), 428–445 (2003)
8. Liu, R., Bhattacharya, K., Wu, F.Y.: Modeling business contexture and behavior using business artifacts. In: Krogstie, J., Opdahl, A.L., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 324–339. Springer, Heidelberg (2007)
9. Gereide, C.E., Su, J.: Specification and Verification of Artifact Behaviours in Business Process Models. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 181–192. Springer, Heidelberg (2007)
10. Hull, R.: Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part II. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008)
11. Bhattacharya, K., Gereide, C., Hull, R.: Towards Formal Analysis of Artifact-Centric Business Process Models. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 288–304. Springer, Heidelberg (2007)
12. Küster, J., Ryndina, K., Gall, H.: Generation of Business Process Models for Object Life Cycle Compliance. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 165–181. Springer, Heidelberg (2007)