Behavioral Conformance of Artifact-Centric Process Models

Dirk Fahland, Massimiliano de Leoni, Boudewijn F. van Dongen, and Wil M.P. van der Aalst

Eindhoven University of Technology, The Netherlands
(d.fahland|m.d.leoni|b.f.v.dongen|w.m.p.v.d.aalst)@tue.nl

Abstract. The use of process models in business information systems for analysis, execution, and improvement of processes assumes that the models describe reality. *Conformance checking* is a technique to validate how good a given process model describes recorded executions of the actual process. Recently, *artifacts* have been proposed as a paradigm to capture dynamic, and inter-organizational processes in a more natural way. In *artifact-centric processes*, several restrictions and assumptions of classical processes are dropped. This renders checking their conformance a more general problem. In this paper, we study the conformance problem of such processes. We show how to partition the problem into *behavioral conformance* of single artifacts and interaction conformance between artifacts, and solve behavioral conformance by a reduction to existing techniques.

Keywords: artifacts, process models, conformance

1 Introduction

Process models have become an integral part of modern information systems where they are used to document, execute, monitor, and optimize processes. However, many studies show that models often deviate from reality (see. [1]). Hence, before a process model can reliably be used, it is important to know in advance to what extent the model conforms to reality.

Classical process modeling techniques assume monolithic processes where process instances can be considered in isolation. However, when looking at the data models of ERP products such as SAP Business Suite, Microsoft Dynamics AX, Oracle E-Business Suite, Exact Globe, Infor ERP, and Oracle JD Edwards EnterpriseOne, one can easily see that this assumption is *not* valid for real-life processes. There are one-to-many and many-to-many relationships between data *objects*, such as customers, orderlines, orders, deliveries, payments, etc. For example, an online shop may split its customers' *quotes* into several *orders*, one per supplier of the quoted items, s.t. each order contains items *for several customers*. Consequently, several customer cases synchronize on the same order at a supplier, and several supplier cases synchronize on the same quote of a customer. In consequence, we will not be able to identify a unique notion of a *process instance* by which we can trace and isolate executions of such a process, and classical modeling languages are no longer applicable [2–4].

The fabric of real-life processes cannot be straightjacketed into monolithic processes. Therefore, we need to address two problems:

(1) Find a modeling language $\mathcal{L}$ to express process executions where several cases of different objects overlap and synchronize;

(2) The *conformance checking problem*: determine whether a process model M expressed in $\mathcal{L}$ adequately describes actual executions of a dynamic and inter-organizational processes in reality — despite the absence of process instances.

The first problem is well-known [2–4] and several modeling languages have been proposed to solve it culminating in the stream of *artifact-centric process modeling* [2–6]. An *artifact instance* is an object that participates in the process. An *artifact* describes a class of similar objects, e.g., all *orders*, together with the *life cycle* of states and possible transitions that each of these objects follows in a process execution. An artifact-centric process model then describes how several artifact instances interact with each other in their respective life cycles. In this paper, we use *proclets* [2] to describe artifact-centric process models and to study and solve the second problem of conformance checking.

Conformance checking compares the behavior described by a process model M to process executions in an actual information system S. Classically S records all events of one execution in an isolated *case*; all cases together form a *log*. Existing conformance checking techniques then check to which degree a given process model can replay each case in the log [7–12]. Artifact-centric systems drop the assumption of an isolated case and a log. Here, S records events in a *database* $\mathcal{D}$ [4]. Each event stored in $\mathcal{D}$ is associated to a unique artifact instance. A complete case follows from an interplay of several artifact instances and several cases overlap on the same artifact instance. Existing conformance checkers cannot be applied in this setting.

In this paper, we investigate the conformance checking problem of artifacts. The problem decomposes into subproblems of significantly smaller size which we reduce to classical conformance checking problems. We contribute a technique to extract logs $L_1, \ldots, L_n$ of logs from a given database, one log for each artifact in the model. Each case of L_i contains all events associated to a specific instance of artifact i. Feeding $L_1, \ldots, L_n$ into existing conformance checkers [12] allows to check conformance of an artifact-centric process model w.r.t. artifact life-cycles as well as artifact interactions.

The paper is structured as follows. Section 2 presents the artifact-centric approach and *proclets* [2] as a light-weight formal model for artifacts. In Sect. 3, we and state the *artifact conformance problem*. Section 4 introduces our techniques for reducing behavioral conformance and interaction conformance to classical process conformance; these techniques and conformance checkers are implemented in the Process Mining Toolkit ProM (available at www.processmining.org). The paper concludes with a discussion on related and future work.

2 The Artifact-Centric Approach

Artifacts emerged in the last years as an alternative approach for precisely describing dynamic, inter-organizational processes in a modular way [3–6]. In the following, we recall the key concepts of artifacts and present a simple formal model for artifact-centric processes that we will use in this paper.

Data objects and artifacts. Artifacts compose complex processes from small building blocks [3, 4]. The particular feature of artifacts is their foundation in the process' under-

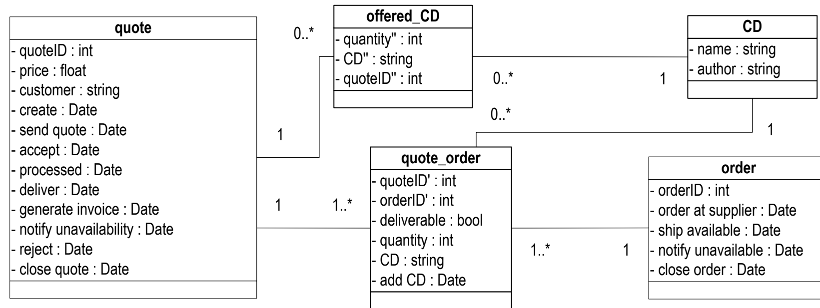

Fig. 1. Data model of a CD online shop's back-end process.

lying *data model*. The approach assumes that any process materializes itself in the (data) objects that are involved in the process, for instance, a paper form, a CD, a customer's quote, or an electronic order; these objects have properties such as the values of the fields of a paper form, the processing state of an order, or the location of a package.

A data model describes the (1) *classes* of objects that are relevant in the process, (2) the relevant properties of these objects in terms of class *attributes*, and (3) the *relations* between the classes. A process execution *instantiates* new objects and changes their properties according to the process logic. Thereby, the relations between classes describe how many objects of one class are related to how many objects of another class.

An *artifact-centric process model* enriches the classes themselves with process logic restricting how objects may evolve during execution. More precisely, one *artifact* (1) encapsulates several classes of the data model, (2) provides *actions* that can update the classes' attributes, (3) defines a *life cycle*, and (4) exposes some of its actions via an *interface*. The artifact's life cycle describes when an *instance* of the artifact (i.e., a concrete object) is created, in which state of the instance which actions may occur to advance the instance to another state, and which *goal state* the instance has to reach to complete a case.

An example. As a running example for this paper, we consider the backend process of a CD online shop. The shop offers a large collection of CDs from different suppliers to its customers. The backend process is triggered by a customer's request for CDs. The shop then sends a *quote* of the offered CDs. If the customer accepts, the quote is split into several *orders*, one per CD supplier. Each order in turn handles all quotes for CDs from the same supplier. The order then is executed and the suppliers ship the CDs to the shop which distributes the different CDs from the different orders according to the original quotes. Some CDs may be unavailable at the supplier; in this case notifications are sent to the CD shop which forwards it to the customer. From an artifact perspective, this backend process is driven by the *quotes* and *orders*, their respective processing states, and their relations. The UML class diagram of Fig. 1 denotes the data model of our CD shop example.

Describing processes by proclet systems. Proclets propose concepts for describing artifacts and their interactions [2]. A *proclet* $P = (N, ports)$ consists of a labeled Petri net, which describes the internal life cycle of one artifact, and a set of ports, through which P can communicate with other proclets [13]. Relations between several proclets

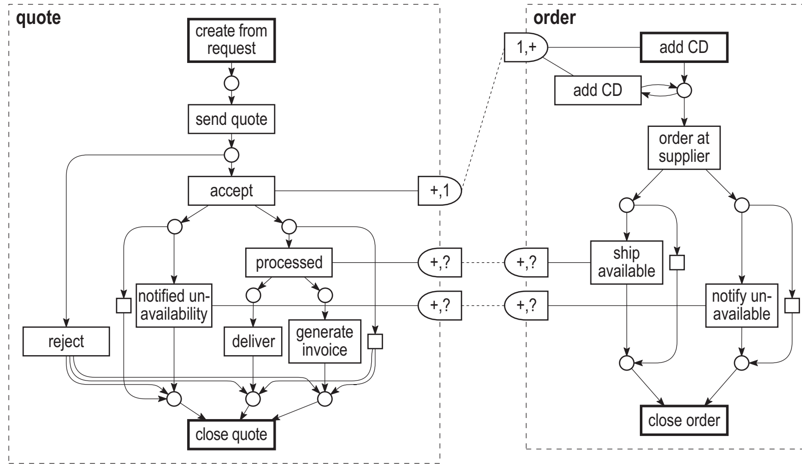

Fig. 2. A proclet system describing the back-end process of a CD online shop. A customer's quote is split into several orders according to the suppliers of the CDs; an order at a supplier handles several quotes from different customers.

are described in a *proclet system* $\mathcal{P} = (\{P_1, \ldots, P_n\}, C)$ consisting of a set of proclets $\{P_1, \ldots, P_n\}$ and a set C of *channels*. Each channel $(p, q) \in C$ connects two ports p and q of two proclets of $\mathcal{P}$. On one hand, proclets *send* and *receive* messages along these channels. On the other hand, the channels also reflects the relations between classes: annotations at the ports define how many *instances* of a proclet interact with how many *instances* of another proclet.

Figure 2 shows a proclet system of two proclets that model artifacts quote and order. Each half-round shape represents a port: the bow indicates the direction of communication. A dashed line between 2 ports denotes a channel of the system. Creation and termination of an artifact instance is expressed by a respective transition, drawn in bold lines in Fig. 2. Note that other modeling languages are likewise applicable to describe an artifact's life cycle [3–6]. Proclets can be mapped to the data model of the process: for each proclet transition (e.g., add quote) exists a corresponding timestamp attribute that is set when the transition occurs (e.g., add quote of quote_order).

The decisive expressivity of proclets for describing artifacts comes from the annotations $1, ?, +$ that are inscribed in the ports [2]. The first annotation, called *cardinality*, specifies how many messages one proclet instance sends to (receives from) other instances when the attached transition occurs. The second annotation, called *multiplicity*, specifies how often this port is used in the lifetime of a proclet instance. For example, the port of accept has cardinality + and multiplicity 1 denoting that a quote once sends out one or more messages on quoted CDs to multiple orders. Conversely, the process repeatedly (+) adds one CD of a quote to an order. These constraints reflect the relation 1..*-1..* between quotes and orders denoted in Fig. 1.

The *semantics* of proclets generalizes the semantics of Petri nets by the ports. Basically, different proclet instances are distinguished by using instance identifiers as tokens. A transition at an output port *produces* as many messages (to other proclet instances) into the channel as specified by the port's cardinality. A transition at an input port waits

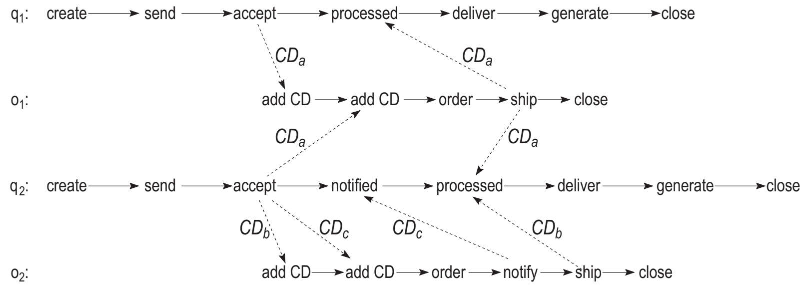

Fig. 3. An execution of the proclet system of Fig. 2 with two quote instances and two order instances.

quote

quoteID	create	send quote	accept	processed	notify	deliver	generate	reject	close quote
q_1	24-11,17:12	24-11,17:13	25-11,7:20	5-12,9:34	null	6-12,5:23	null	null	6-12,5:25
q_2	24-11,19:56	24-11,19:57	25-11,8:53	5-12,11:50	3-12,14:54	6-12,7:14	3-12,14:55	null	6-12,7:20

order

orderID	ship	order	notify	close order
o_1	5-12,9:32	28-11,8:12	null	5-12,11:37
o_2	5-12,11:33	28-11,12:22	3-12,14:34	5-12,13:03

CD

name	author
a	xyz
b	zyx
c	yxz

offered_CDs

quoteID"	CD"	quantity"
q_1	a	2
q_2	a	1
q_2	b	3
q_2	c	1

quote_order

quoteID'	orderID'	add CD	deliverable	CD
q_1	o_1	25-11,8:31	true	a
q_2	o_1	25-11,12:11	true	a
q_2	o_2	26-11,9:30	true	b
q_2	o_2	26-11,9:31	false	c

Fig. 4. Events of the run of Fig. 3 recorded in a database according to Fig. 1.

for as many messages (from other proclet instances) as specified and consumes them. For example, Fig. 3 illustrates an execution of the proclet system of Fig. 2: one over $\mathsf{CD_a}$ and the other over $\mathsf{CD_a}$, $\mathsf{CD_b}$, and $\mathsf{CD_c}$. $\mathsf{CD_b}$ and $\mathsf{CD_c}$ have the same supplier, $\mathsf{CD_a}$ has a different supplier. Hence, the quotes are split into two orders. In the run, $\mathsf{CD_a}$ and $\mathsf{CD_b}$ are available whereas $\mathsf{CD_c}$ is not, which leads to the behavior shown in Fig. 3 involving two quote instances and two order instances.

Operational semantics of proclets specify senders of messages to consume and recipients of produced messages [2]. For conformance checking, focusing on the number of produced and consumed messages is sufficient; see [13] for details. For example the run of Fig. 3 satisfies all cardinality and multiplicity constraints of the ports of Fig. 2, i.e., it *conforms* to the proclet system. A system that executes this process records timestamps of events in a database according to the data model of Fig. 1. The corresponding database tables could be populated as shown in Fig. 4. The question that we consider in the following is whether the model of Fig. 2 accurately describes the records of Fig. 4.

3 The Artifact Conformance Checking Problem

The problem of determining how accurately a process model describes the process implemented in an actual information system S is called *conformance checking problem* [7].

Classically, a system S executes a process in an isolated instance. The corresponding *observed* system execution is a sequence of events, called *case*, and a set of cases is a *log*

L. The semantics of a formal process model M define the set of valid process executions in terms of sequences of M's actions. Conformance of M to L can be characterized in several dimensions [7]. In the following, we consider only *fitness*. This is the most dominant conformance metric that describes to which degree a model M can replay all cases of a given log L, e.g., [12]. M fits L less, for instance, if M executes some actions in a different order than observed in L, or if L contains actions not described in M. Several conformance checking techniques for process models are available [7–12]. The more robust techniques, e.g., [12], find for each case $\rho \in L$ an execution ρ' of M that is as *similar* as possible to ρ; the similarity of all ρ to their respective ρ' defines the fitness of M to L.

3.1 The Artifact Conformance Problem

We have seen in Sections 1 and 2 that many processes do not structure their executions into isolated instances. In the light of this observation, we identify the following *artifact conformance problem*. The system S records occurrences of a set Σ of actions in a *database* $\mathcal{D}$ according to the system's data model. Each event is associated to a specific object, that is stored in $\mathcal{D}$. Let $\mathcal{P}$ be a proclet system where each proclet transition maps to a timestamped attribute of $\mathcal{D}$ (i.e., each proclet of $\mathcal{P}$ describes an artifact of S). Can the proclets of $\mathcal{P}$ be instantiated s.t. the life-cycles of all artifact instances and their interaction "replay" *all* events recorded in $\mathcal{D}$? If not, to which degree does $\mathcal{P}$ deviate from the behavior recorded in $\mathcal{D}$?

3.2 Reducing Artifact Conformance to Existing Techniques

A naïve solution of the artifact conformance problem would replay all events of the database $\mathcal{D}$ in the proclet system $\mathcal{P}$. Technically, this would mean to find the database $\mathcal{D}'$ that can be replayed by $\mathcal{P}$ and is as similar as possible to $\mathcal{D}$. In typical case studies we found the actual system S to record about 80,000 events of 40-60 distinct actions. Finding a conforming database $\mathcal{D}'$ by replacing non-conforming events with conforming events defines a search space of $80,000^{60}$ possible solutions. Even exploring only a small fraction of such a search space quickly turns out infeasible.

For this reason, we propose a compositional approach to check whether an proclet system $\mathcal{P}$ fits $\mathcal{D}$. As we cannot employ the notion of a process instance to structure $\mathcal{D}$ into smaller parts we partition the problem into checking conformance *within* proclets and *between* proclets.

Behavioral conformance. Each event in $\mathcal{D}$ is associated to an object, and hence to an instance i of an artifact Ar described by a proclet P_{Ar} in the proclet system $\mathcal{P}$. All events associated to i together constitute the *artifact case* of i of Ar that describes how i evolved along the life-cycle of Ar. It ignores how i interacts with other artifact instances. The *behavioral conformance* problem is to check whether the life cycle of P_{Ar} can replay each artifact case of Ar (i.e., each recorded artifact life cycle).

Interaction conformance. Completing a life cycle of an instance i of Ar also depends on other artifact instances, as discussed previously. Let J be the set of artifact instances with which i exchanges messages. All events of $\mathcal{D}$ that send or receive messages and

are associated to an instance in $\{i\} \cup J$ together constitute the *interaction case* of Ar. It contains all behavioral information regarding how i interacts with other instances. Proclet P_{Ar} *fits* the interaction case of instance i of Ar if the interaction case involves events of as many artifact instances as required by the ports of P_{Ar}. The *interaction conformance* problem is to check how good all proclets of $\mathcal{P}$ fits all interactions cases that are stored in $\mathcal{D}$; it describes how good the proclet interactions reflect the object relations in $\mathcal{D}$.

The behavioral conformance and the interaction conformance together yield the artifact conformance of the entire proclet system $\mathcal{P}$ w.r.t. $\mathcal{D}$; see [13] for a formal proof. Yet, either conformance can be checked per artifact case or per interaction action case, respectively, which significantly reduces the search space during checking.

4 Checking Behavioral Conformance of Artifacts

In the following, we first solve the *behavioral conformance problem* by reduction to classical process conformance. Assuming that events of artifacts $Ar_1, \dots, Ar_n$ are recorded in a given database $\mathcal{D}$, we extract for each artifact Ar_i all artifact cases from $\mathcal{D}$ into a log L_i. The logs $L_1, \dots, L_n$ describe the internal life cycle behavior of the artifacts. These logs can then be used to check behavioral conformance of a proclet system w.r.t. $\mathcal{D}$ in existing conformance checkers, as we show in Sect. 4.3. Moreover, the logs $L_1, \dots, L_n$ can be leveraged to also express interaction between artifacts, which then allows to check interaction conformance with existing conformance checkers [13].

4.1 Extracting logs from databases

In the following, we provide a technique to extract logs from a relational database $\mathcal{D}$. We assume that $\mathcal{D}$ recorded events of n different artifacts, and that each event is associated to a specific instance of an artifact. Our vehicle to extract logs from $\mathcal{D}$ will be an *artifact view* on $\mathcal{D}$ which specifies for each artifact of the system, the *types* of events occurring in this artifact. Each event type is characterized in terms of database attributes (of different tables) of $\mathcal{D}$ which need to be related to each other according to the schema of $\mathcal{D}$. Using this characterization, we then extract events from $\mathcal{D}$ by joining tables, and selecting and projecting entries according to the specified attributes. We first introduce some notion on databases and then present the details of this approach.

Preliminaries. We adopt notation from *Relational Algebra* [14]. A *table* $T \subseteq D_1 \times \ldots \times D_m$ is a relation over domains D_i and has a *schema* $\mathcal{S}(T) = (A_1, \dots, A_m)$ defining for each *column* $1 \leq i \leq m$ an *attribute name* A_i. For each *entry* $t = (d_1, \dots, d_m) \in T$ and each column $1 \leq i \leq m$, let $t.A_i := d_i$. We write $\mathcal{A}(T) := \{A_1, \dots, A_m\}$ for the attributes of T, and for a set $\mathcal{T}$ of tables, $\mathcal{A}(\mathcal{T}) := \bigcup_{T \in \mathcal{T}} \mathcal{A}(T)$. A *database* $\mathcal{D} = (\mathcal{T}, K)$ is set $\mathcal{T}$ of tables with corresponding schemata $\mathcal{S}(T), T \in \mathcal{T}$ s.t. their attributes are pairwise disjoint, and a *key relation* $K \subseteq (\mathcal{A}(\mathcal{T}) \times \mathcal{A}(\mathcal{T}))^{\mathbb{N}}$.

K expresses foreign-primary key relationships between the tables $\mathcal{T}$: we say that $\big((A_1, A'_1), \dots, (A_k, A'_k)\big) \in K$ *relates* $T \in \mathcal{T}$ to $T' \in \mathcal{T}$ iff the attributes $A_1, \dots, A_k \in \mathcal{A}(T)$ together are a *foreign key* of T pointing to the *primary key* $A'_1, \dots, A'_k \in \mathcal{A}(T')$

of T'. For instance, $(\mathsf{quoteID}, \mathsf{quoteID}')$ is a foreign-primary key relation from table quote to table quote_order of Fig. 4.

Relational algebra defines several operators [14] on tables. In the following, we use *projection*, *selection*, and the canonical crossproduct. For a table T and attributes $\{A_1, \ldots, A_k\} \subseteq \mathcal{A}(T)$, the projection $Proj_{A_1,\ldots,A_k} T$ restricts each entry $t \in T$ to the columns of the given attributes $A_1, \ldots, A_k$. Please note that projection removes any duplicates: if there are two entries in $t_1, t_2 \in T$ that coincide on the values of the projected attributes $A_1, \ldots, A_k$ (i.e., $t_1.A_1 = t_2.A_2 \wedge t_1.A_n = t_2.A_n$), after projecting, the entry obtained by projection t_2 is removed. Selection is a unary operation $Sel_\varphi(T)$ where φ is a boolean formula over atomic propositions $A = c$ and $A = A'$ where $A, A' \in \mathcal{A}(T)$ and c a constant; the result contains entry $t \in Sel_\varphi(T)$ iff $t \in T$ and t satisfies φ (as usual). We assume that each operation correspondingly produces the schema $\mathcal{S}(T')$ of the resulting table T'.

For a set $\mathcal{T}' = \{T_1, \ldots, T_k\} \subseteq \mathcal{T}$ of tables, let $J_{K,\mathcal{T}'} := \{(A, A') \in k \mid k \in K, A, A' \in \mathcal{A}(\mathcal{T}')\}$ denote the pairs of attributes that are involved in key relations between the tables in $\mathcal{T}'$. The $Join(\mathcal{T}', K) := Sel_\varphi(T_1 \times \ldots \times T_k)$ with $\varphi := \bigwedge_{(A_i,A') \in J_{K,\mathcal{T}'}} (A_i = A'_i)$ keeps from the cross-product of all tables $\mathcal{T}'$ only those entries which coincide on all key relations.

With these notions at hand, we first introduce an *artifact view* on a database $\mathcal{D}$. It specifies for each artifact the types of events that are recorded in $\mathcal{D}$. Each event type is characterized by attributes of the database, defining in which instance an event occurred and when it occurred. We later use an artifact view to extract all events of an artifact and group them into cases, which yields a log.

Definition 1 (Artifact View). *Let $\mathcal{D} = (\mathcal{T}, K)$ be a database. An* artifact view $V = (\{\Sigma_1, \ldots, \Sigma_n\}, Tab, Inst, TS)$ *on $\mathcal{D}$ is specified as follows:*

- *It defines n pairwise disjoint sets $\Sigma_1, \ldots, \Sigma_n$ of* event types *(one set per artifact). Let $\Sigma := \bigcup_{i=1}^{n} \Sigma_i$.*
- *Function $Tab : \{\Sigma_1, \ldots, \Sigma_n\} \to 2^{\mathcal{T}}$ specifies the set $Tab(\Sigma_i)$ of tables linked to each artifact $i = 1, \ldots, n$.*
- *Function $Inst : \{\Sigma_1, \ldots, \Sigma_n\} \to \mathcal{A}(\mathcal{T})$ specifies for each each artifact $i = 1, \ldots, n$ the attribute $Inst(\Sigma_i) = A^i_{id} \in \mathcal{A}(Tab(\Sigma_i))$ that uniquely identifies an instance of this artifact.*
- *Function $TS : \Sigma \to \mathcal{A}(\mathcal{T})$ specifies for each event type $a \in \Sigma$ the* timestamp *attribute $TS(a) = \mathcal{A}_{TS} \in \mathcal{A}(Tab(\Sigma_i))$ that records when an event of type a occurred. Attributes $Inst(\Sigma_i)$ and $TS(a)$ must be connected through tables $\mathcal{T}' \subseteq Tab(\Sigma_i)$.*

$Tab(\Sigma_{\mathsf{order}}) = \{\mathsf{quote_order}, \mathsf{order}\}$,
$Inst(\Sigma_{\mathsf{order}}) = \mathsf{orderID}$

event type $a \in \Sigma_{\mathsf{order}}$	$TS(a)$
add CD	add CD
order at supplier	order
ship available	ship
notify unavailable	notify
close order	close order

Table 1. *Artifact view* for order

Table 1 presents the artifact view for the artifact order of our running example on the database of Fig 4. The choice of the event types Σ_{order}, tables $Tab(\Sigma_{\mathsf{order}})$, the instance identifier orderID and the corresponding time stamp attributes is straight forward.

After specifying an artifact view, an artifact log can be extracted fully automatically from a given database $\mathcal{D}$.

Definition 2 (Log Extraction). *Let $\mathcal{D} = (\mathcal{T}, K)$ be a database, let $V = (\{\Sigma_1, \ldots, \Sigma_n\}, Tab, Inst, TS)$ be an artifact view on $\mathcal{D}$. The logs $L_1, \ldots, L_n$ are extracted from $\mathcal{D}$ as follows. For each set $\Sigma_i, i = 1, \ldots, n$ of event types:*

1. *Each event type $a \in \Sigma_i$ defines the event table $T_a = Proj_{Inst(\Sigma_i), TS(a)} Join(Tab(\Sigma_i), K)$.*
2. *Each entry $t = (id, ts) \in T_a$ identifies an event $e = (a, id, ts)$ of type a in instance id. Let $\mathcal{E}_i$ be the set of all events of all event types $a \in \Sigma_i$.*
3. *For each instance $id \in \{id \mid (a, ts, id) \in T_a, a \in \Sigma_i\}$ of artifact $i \in \{1, \ldots, n\}$, the set $\mathcal{E}_i|_{id} = \{(a, ts, id') \in \mathcal{E} \mid id = id'\}$ contains all events of instance id.*
4. *The* artifact case *$\rho_{id} = \langle a_1, a_2, \ldots a_n \rangle$ of instance id of artifact i orders events $\mathcal{E}|_{id}$ by their timestamp: $\mathcal{E}_i|_{id} = \{(a_1, id, ts_1), (a_2, id, ts_2), \ldots, (a_n, id, ts_n)\}$ s.t. $ts_i < ts_{i+1}$, for all $1 \leq i < n$. The log L_i contains all artifact cases of artifact i.*

quoteID'	...	orderID	add CD	ship	...
q_1	...	o_1	25-11,8:31	5-12,9:32	...
q_2	...	o_1	25-11,12:11	5-12,9:32	...
q_2	...	o_2	26-11,9:30	5-12,11:33	...
q_2	...	o_2	26-11,9:31	5-12,11:33	...

Table 2. Intermediate table obtained by joining $Join(\{\text{quote_order}, \text{order}\}, K)$.

We illustrate the log extraction by our running example from Sect. 2. For the database of Fig. 4, we consider the artifact view on order as specified in Tab. 1. To extract events of order first join the tables order and quote_order on (orderID, orderID′), Tab. 2 shows parts of that table. To obtain events of type add CD, project this tables onto $Inst(\Sigma_{\text{order}}) = \text{orderID}$ and timestamp attribute $TS(\text{add CD}) = \text{add CD}$, which yields four entries $(o_1, \text{25-11,8:31})$, $(o_1, \text{25-11,12:11})$, $(o_2, \text{26-11,9:30})$, and $(o_2, \text{26-11,9:31})$. For event ship available, the projection onto $Inst(\Sigma_{\text{order}}) = \text{orderID}$ and $TS(\text{ship available}) = \text{ship}$ yields two entries $(o_1, \text{5-12,9:32})$ and $(o_2, \text{5-12,11:33})$, duplicates are removed. Extracting all other events and grouping them by orderID yields two cases: $\rho_{o_1} = \langle \text{add CD}, \text{add CD}, \text{order}, \text{ship}, \text{close} \rangle$ and $\rho_{o2} = \langle \text{add CD}, \text{add CD}, \text{order}, \text{notify}, \text{ship}, \text{close} \rangle$.

4.2 Checking Behavioral Conformance of Artifacts with Existing Techniques

With the notions of an artifact view (Def. 1) and automatic log extraction (Def. 2), we reduced the behavioral conformance problem to a classical setting: behavioral conformance of artifacts can be checked using existing conformance checkers.

Given a database $\mathcal{D}$ and a proclet system $\mathcal{P} = (\{P_1, \ldots, P_n\}, C)$ where each proclet P_i describes an artifact of the system, first define an artifact viewpoint V that specifies for each proclet P_i and each transition label a in P_i an event type $a \in \Sigma_i$ in terms of $\mathcal{D}$. Then extract the artifact logs $L_1, \ldots, L_n$ from $\mathcal{D}$ using V.

Then check behavioral conformance of each proclet P_i w.r.t. $\mathcal{D}$ by checking conformance of the Petri net that underlies P_i w.r.t. the log L_i, by ignoring the ports of P_i. A corresponding conformance checker [12] tries to replay each case ρ in L_i by firing transitions of P_i in the order given in ρ. If a transition cannot be fired, the checker searches for a log ρ' that is as similar to ρ as possible and that can be replayed in P_i. We implemented this approach: logs can be extracted using XESame [15], the Process Mining Toolkit ProM checks conformance of a proclet system and provides diagnostics on non-conformance per artifact case (Fig. 5).

The life cycle model of order of Fig. 2 conforms to the log L_{order} extracted from the database of Fig. 4, i.e., the two traces just presented. The conformance checker [12] will also report for orderID $= o_1$ that an "unobservable activity" occurred (to bypass notify). The cases for quote of Fig. 4 stored in Fig. 4 yield a different result. Here, the trace of quoteID $= q_1$ lacks an event for generate invoice and an "activity in the model that was not logged" is reported. The trace of quoteID $= q_2$ generates an invoice before the order is processed, so an "activity of the log that was not (yet) enabled" is reported.

4.3 Checking Interaction Conformance of Artifacts

We just showed how to check behavioral conformance artifacts, i.e., whether the internal life cycles of each artifact, described by a proclet, conform to the artifact cases stored in a database $\mathcal{D}$. Complete artifact conformance also requires to check conformance w.r.t. interactions between proclets. In the following, we sketch how to leverage the notions of a viewpoint (Def. 1) and of log extraction (Def. 2) to extract so called *instance aware logs*. Using instance aware logs, interaction conformance of artifacts can be checked again using existing techniques [13].

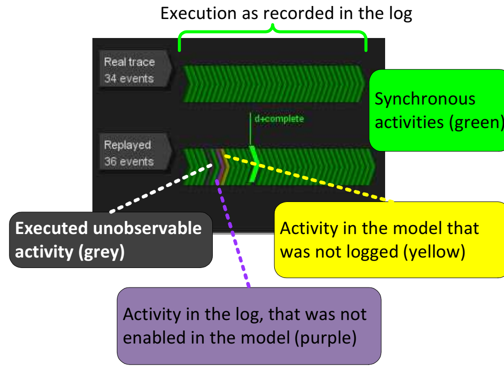

Fig. 5. Screenshot of ProM, showing conformance results of the proclet system of Fig. 2.

In an *instance-aware log*, an event $e = (a, id, SID, RID)$ not only describes that an event of type a occurred in instance id; it also describes from which instances SID the event *consumed* a message, and for which instances RID the event *produced* a message. For instance, the *instance-aware cases* of artifact order of Fig. 3 are

$$\rho_{o1} : \langle(\text{add CD}, o_1, [q_1], []), (\text{add CD}, o_1, [q_2], []), (\text{order at supplier}, o_1, [], []), \\ (\text{ship available}, o_1, [], [q_1, q_2]), (\text{close order}, o_1, [], [])\rangle$$
$$\rho_{o2} : \langle(\text{add CD}, o_2, [q_2], []), (\text{add CD}, o_2, [q_2], []), (\text{order at supplier}, o_2, [], []), \\ (\text{notify unavailable}, o_2, [], [q_2]), (\text{ship available}, o_2, [], [q_2]), (\text{close order}, o_1, [], [])\rangle$$

This information suffices to enrich each instance-aware case of an instance i with those events that produced a message for i or consumed a message from i. The resulting cases equivalently capture the interaction behavior that is stored in $\mathcal{D}$, and they can be fed to existing conformance checkers [13]. To extract SID from $\mathcal{D}$, the artifact view (Def. 1) needs to be extended.

Events of type $a \in \Sigma_i$ may consume messages that were produced by a specific artifact. The attribute A^s_{id} that distinguishes the different instances of that artifact must be specified. The instance identifier $Inst(\Sigma_i)$ of the artifact of $a \in \Sigma_i$ and A^s_{id} must be connected by tables $\mathcal{T}'$ of $\mathcal{D}$. Not every connection between $Inst(\Sigma_i)$ and A^s_{id} implies that a message was exchanged; a guard g over $\mathcal{T}'$ specifies when this is the case. For instance, the set SID of ship available contains all identifiers of attribute quoteID when the guard deliverable $=$ true evaluates to true.

The log extraction (Def. 2) needs to be extended correspondingly. For each entry $t = (id, ts) \in T_a$ in the event table of event type $a \in \Sigma_i$, extract values for SID as follows: joining all tables that connect attributes $Inst(\Sigma_i)$ and A^s_{id}, select from the result only the entries which satisfy $Inst(\Sigma_i) = id \wedge TS(a) = \wedge g$ (i.e., entries referring to t where also the guard g holds), and project the result onto A^s_{id}. The set RID of instances for which e produced a message is specified and extracted likewise. This procedure yields instance-aware logs $L_1, \ldots, L_n$, one for each artifact in $\mathcal{D}$.

5 Conclusion

In this paper, we considered the problem of checking how a given process model *conforms* to executions of the actual process — under the realistic assumption that process executions are *not* structured into monolithic process instances. Rather, executions of most processes in reality are driven by their data objects which may participate in various, overlapping *cases*. Usually, the life cycle history of each objects that is involved in a process execution is recorded in a structured database. Likewise, the objects, their life cycles, and their interactions can be expressed in an artifact-centric process models, for instance using *proclets* [2].

In this setting, the *conformance problem* is to check how good a given proclet system describes all events recorded in the database. We decomposed this conformance problem in Sect. 3 into (1) the *behavioral conformance problem* on how good a proclet describes events of an artifact instance, and (2) the *interaction conformance problem* on how good the proclet system artifact interactions. Section 4 reduced behavioral conformance to classical conformance by extracting a classical process log for each artifact life cycle from the given database; technically, the log follows from a view on the database. The technique is likewise applicable for checking interaction conformance [13]; it is implemented in the Process Mining Toolkit ProM.

Related Work. Conformance checking, that is, comparing formal process models to actual process executions is a relatively new field that was studied first on monolithic processes with isolated process instances [16]. To the best of our knowledge, the conformance problem has not been studied yet for artifact-centric processes. Our approach currently only reduces artifact conformance to classical conformance. Yet, classical conformance checking knows several metrics which describe conformance differently [16].

The most advanced conformance metrics reflect that only parts of a trace are deviating [10, 17], and pinpoint where deviations occur [11], while taking into account that models may contain behavior that is unobservable by nature [12]. In particular the last metric can be applied to several process modeling languages, including proclets used in Sect. 2 to describe artifacts.

Open Issues. This paper made a first step towards checking conformance of artifact-centric process models. Currently, we manually have to specify the artifact view on the database by identifying which tables relate to which artifact, and which attributes relate to which event. This can be cumbersome, as the relations between tables (expressed by foreign-primary key relations) need to be respected. A view is insensitive to adding further tables or attributes to the database, but sensitive to changes in the key relations. For this reason, automated techniques for checking *structural conformance* of a given

proclet system to a database, and for *discovering* conformant artifact views for a given proclet system from a database would be required. Furthermore, metrics such as [12] need to be adapted to the artifact setting to describe the *degree* to which a process model describes observed executions. Finally, as artifact-centric processes are data-driven, also conformance of data-dependent guards to recorded process executions is an open issue.

Acknowledgements. The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement no. 257593 (ACSI).

References

1. Rozinat, A., Jong, I., Gunther, C., Aalst, W.: Conformance Analysis of ASML's Test Process. In: GRCIS'09. Volume 459 of CEUR-WS.org. (2009) 1–15
2. Aalst, W., Barthelmess, P., Ellis, C., Wainer, J.: Proclets: A Framework for Lightweight Interacting Workflow Processes. Int. J. Cooperative Inf. Syst. **10** (2001) 443–481
3. Nigam, A., Caswell, N.: Business artifacts: An approach to operational specification. IBM Systems Journal **42** (2003) 428–445
4. Cohn, D., Hull, R.: Business artifacts: A data-centric approach to modeling business operations and processes. IEEE Data Eng. Bull. **32** (2009) 3–9
5. Fritz, C., Hull, R., Su, J.: Automatic construction of simple artifact-based business processes. In: ICDT'09. Volume 361 of ACM ICPS. (2009) 225–238
6. Lohmann, N., Wolf, K.: Artifact-centric choreographies. In: ICSOC 2010. Volume 6470 of LNCS., Springer (2010) 32–46
7. Rozinat, A., de Medeiros, A.K.A., Günther, C.W., Weijters, A.J.M.M., van der Aalst, W.M.P.: The Need for a Process Mining Evaluation Framework in Research and Practice. In: BPM'07 Workshops. Volume 4928 of LNCS., Springer (2007) 84–89
8. Greco, G., Guzzo, A., Pontieri, L., Sacca, D.: Discovering Expressive Process Models by Clustering Log Traces. IEEE Trans. on Knowl. and Data Eng. **18** (2006) 1010–1027
9. Weijters, A., van der Aalst, W.: Rediscovering Workflow Models from Event-Based Data using Little Thumb. Integrated Computer-Aided Engineering **10** (2003) 151–162
10. Medeiros, A., Weijters, A., van der Aalst, W.: Genetic Process Mining: An Experimental Evaluation. Data Mining and Knowledge Discovery **14** (2007) 245–304
11. Rozinat, A., van der Aalst, W.: Conformance Checking of Processes Based on Monitoring Real Behavior. Information Systems **33** (2008) 64–95
12. Adriansyah, A., Dongen, B., Aalst, W.: Towards Robust Conformance Checking. In: BPM'10 Workshops. (2010) LNBIP to appear.
13. Fahland, D., de Leoni, M., van Dongen, B., van der Aalst, W.: Checking behavioral conformance of artifacts. BPM Center Report BPM-11-08, BPMcenter.org (2011)
14. Silberschatz, A., Korth, H.F., Sudarshan, S.: Database System Concepts, 4th Edition. McGraw-Hill Book Company (2001)
15. Verbeek, H., Buijs, J.C., van Dongen, B.F., van der Aalst, W.M.P.: ProM: The Process Mining Toolkit. In: BPM Demos 2010. Volume 615 of CEUR-WS. (2010)
16. Rozinat, A., de Medeiros, A.A., Günther, C., Weijters, A., van der Aalst, W.: Towards an Evaluation Framework for Process Mining Algorithms (2007) BPM Center Report BPM-07-06.
17. Weijters, A., van der Aalst, W., de Medeiros, A.A.: Process Mining with the Heuristics Miner-algorithm. Technical report, Eindhoven University of Technology, Eindhoven (2006) BETA Working Paper Series, WP 166.