Artifact-centric choreographies

Niels Lohmann and Karsten Wolf

Universität Rostock, Institut für Informatik, 18051 Rostock, Germany
{niels.lohmann, karsten.wolf}@uni-rostock.de

Abstract. Classical notations for service collaborations focus either on the control flow of participating services (interacting models) or the order in which messages are exchanged (interaction models). None of these approaches emphasizes the evolution of data involved in the collaboration. In contrast, artifact-centric models pursue the converse strategy and begin with a specification of data objects.
This paper extends existing concepts for artifact-centric business process models with the concepts of *agents* and *locations*. By making explicit *who* is accessing an artifact and *where* the artifact is located, we are able to automatically generate an interaction model that can serve as a contract between the agents and by construction makes sure that specified global goal states on the involved artifacts are reached.

1 Introduction

During the last few years, *artifact-centric* modeling of business processes received more and more attention [21,14,8,10,13,15]. In essence, this paradigm is about modeling the main data objects and then to derive a process which manipulates these data objects for achieving a given business goal. Some authors propose to mirror this paradigm in the implementation of business process engines by proposing an artifact hub which drives the actual business process [15]. Artifact-centric approaches have also been proposed in a service-oriented setting [10,13]. Here, services are discovered and employed for performing actions which manipulate artifacts. Although different in many details, this approach has a bit of an orchestration flavor.

In this paper, we study the use of artifacts in a more choreography-like setting. That is, we abandon the concept of having a single artifact hub. Instead, we introduce the idea of having several *agents* each of which operates only on some of the artifacts in the overall system. We believe that this setting is more appropriate for interorganizational business collaborations than the idea of central artifact hubs.

We observe that some artifacts can move between agents and that their actual location may matter for the executability of actions performed on them. That is, we extend the idea of "moving data throughout a process" [8] by the idea of "moving artifacts between agents". In consequence, we contribute

- a systematic approach for modeling artifact location and its impact on the accessibility of actions, and
- an approach to automatically derive an interaction model between the agents which may serve as a contract between them.

The derivation of the interaction model is solely based on existing methods and tools [18,24,20,19]. Local policies of the involved agents can easily be incorporated.

Models using a single artifact hub appear as a special case in our approach where there is only a single agent. In a setting with more than one agent, however, our approach reaches beyond the single agent setting: We derive not only the actions that need to be performed on the artifacts for reaching a goal but also the messages that need to be exchanged between agents for this purpose.

The paper is organized as follows. The next section briefly describes how artifacts can be modeled with Petri nets. Section 3 introduces location-aware extensions and their impact to artifact models. Section 4 provides a categorization of location information which may serve as suggestion for a high-level language for location-aware artifacts. The construction of an interaction model out of location-aware artifacts is described in Sect. 5. Section 6 shows our approach in the context of related work before Sect. 7 concludes and discusses several possible extensions to the work of this paper.

2 Modeling artifacts

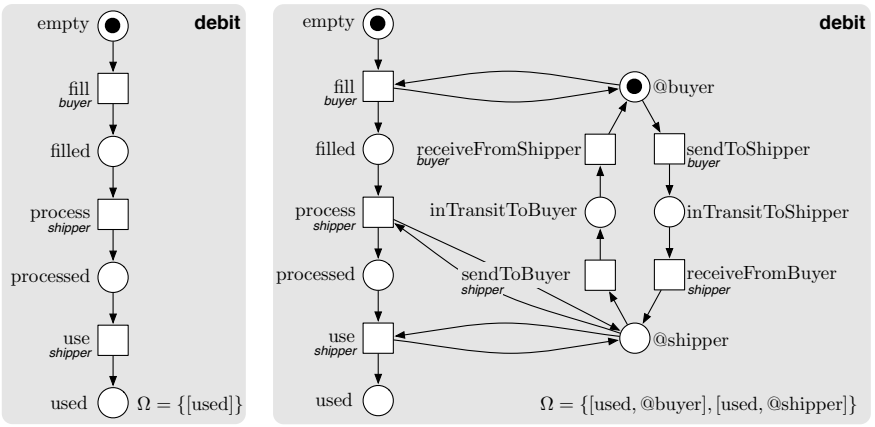
Being concerned with an interorganizational setting, we assume that there is a set $\mathcal{A}$ of *agents* (or roles, organizations, locations, etc.). In our approach, agents are principals in the role-based access control for artifacts. Additionally, agents may (permanently or temporarily) *posses* artifacts, so they play the role of locations that messages containing artifacts may be sent to.

Informally, an artifact consists of data fields that can be manipulated (changed) by agents. Thereby, the change of data is constrained by the role-based access control. Hence, we model an artifact as a state machine. States represent the possible valuations of the data fields whereas transitions are the potential atomic changes that can happen to the data fields. In this paper, we use Petri nets for implicitly representing state machines. When data fields in an artifact evolve independently of each other, the size of a Petri net grows much slower than the number of represented states.

Definition 1 (Petri net). *A* Petri net $N = [P, T, F, m_0]$ *consists of two finite and disjoint sets* P *(*places*) and* T *(*transitions*), a* flow relation $F \subseteq (P \times T) \cup (T \times P)$*, and an* initial marking m_0*. A marking* $m : P \to \mathbb{N}$ *represents a state of the Petri net and is visualized as a distribution of tokens on the places. Transition* t *is enabled in marking* m *iff, for all* $[p, t] \in F$*,* $m(p) > 0$*. An enabled transition* t *can fire, transforming* m *into the new state* m' *with* $m'(p) = m(p) - W(p, t) + W(t, p)$ *where* $W([x, y]) = 1$ *if* $[x, y] \in F$*, and* $W([x, y]) = 0$*, otherwise.*

Transitions are triggered by *actions*. The available actions form the *interface* to the artifact. For modeling role based access control, each action is associated to an agent, meaning that this agent is permitted to perform that action.

In Fig. 1(a), we show an example of an artifact model for a direct debit authorization (automated clearing house). Data fields of such an artifact include an amount authorized, account information, and Booleans showing whether it has been signed, respectively whether it has been validated. In the state machine model, we abstract away the amount and account information fields. These fields do not influence the control flow in our



(a) **debit** artifact (b) location-aware **debit** artifact

Fig. 1. Direct debit authorization artifact (a) with location information (b).

example. We silently assume that each artifact has finitely many states which typically can be achieved by a suitable abstraction.

Throughout this paper fix a set $\mathcal{L}$ of *action labels* and a mapping $c : \mathcal{L} \to \mathcal{A}$ representing the *access control*. In figures, the access control is written as additional transition labels, for instance *buyer* and *shipper* in Fig. 1(a).

Definition 2 (Artifact). *An* artifact $A = [N, \ell, \Omega]$ *consists of*

- *A Petri net* $N = [P, T, F, m_0]$*;*
- *a transition labeling* $\ell : T \to \mathcal{L}$ *associating actions with Petri net transitions;*
- *a set* Ω *of markings of* N *representing endpoints in the life cycle of the artifact.*

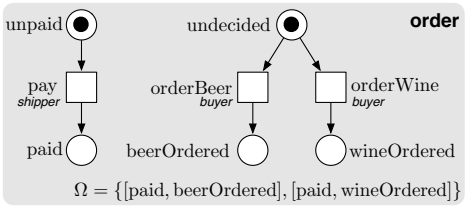
Action $x \in \mathcal{L}$ *is enabled in marking* m *iff some transition* $t \in T$ *with* $l(t) = x$ *is enabled in* m*. Executing the enabled action* x *amounts to firing any (nondeterministically chosen) such transition.*

Nondeterminism in an artifact may sound unusual at first glance but may occur due to prior abstraction. The final marking of the **debit** artifact (cf. Fig. 1(a)) consists of the marking [used] modeling successful debit authorization.

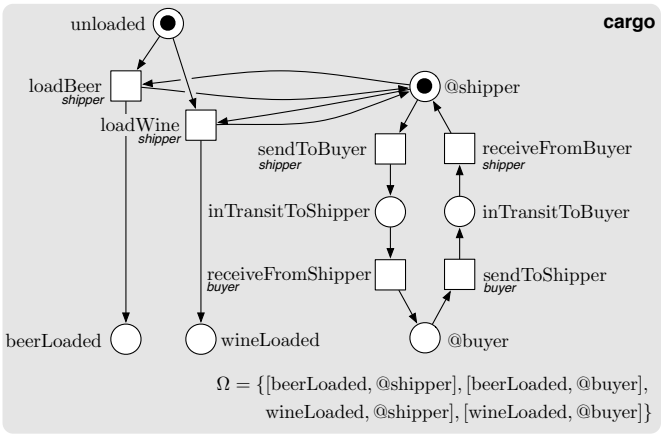
3 Location-aware artifacts

If we want to derive a protocol from a set of given artifacts, we have to understand the reasons for which messages are sent around in an artifact context. It turns out that there exist different shapes of artifacts which cause message transfer for different reasons. We give a few examples.

Consider first an artifact that is materialized as, say, a physical form. Actions in this artifact correspond to filling in fields in this form. Still, some actions may be bound



(a) location-aware **order** artifact



(b) location-aware **cargo** artifact

Fig. 2. Online order artifact (a) and cargo artifact (b).

to particular agents (e.g., signatures), so the artifact itself must be passed to that agent. Passing the artifact corresponds to sending a message. The act of sending would, at the same time, disable any actions by the sending agent. In another scenario, an artifact manifests itself as a database record. In this case, the artifact is not passed, but a message may be required to announce the existence of the artifact and to transmit some kind of access link which then enables the other agent to perform actions on the artifact by remote access to the data base.

Taking the artifact-centric approach seriously, we propose to include the acts of sending a message, receiving a message, and synchronous communication steps as specific actions of the artifact. Likewise, the actual location of the artifact (at an agent or "in transit") becomes an additional data field. The additional data field can be used for modeling the actual effect of the messaging activity such as enabling or disabling other actions. In the example, remote access to the artifact is ruled out. Figure 1(b) shows the augmented model for the direct debit authorization from Fig. 1(a). It expresses the fact that transition "fill" can only be executed when the artifact resides at the buyer, whereas the other actions can only be performed when the artifact is at the shipper. Note that the augmented model is required to leave "in transit" states to reach a final marking.

As a second example, consider an online **order** artifact. Figure 2(a) shows the functional part of such an artifact which happens to be identical to its location-aware

extension. This reflects the idea that the artifact can be remotely accessed at any time. The artifact **cargo** (cf. Fig. 2(b)) is similar to the **debit** artifact and models the fact that the cargo can only be loaded to the shipper.

We see that the extension to the functional artifact model may vary a lot. Hence, it is reasonable to provide this information as part of the artifact. One possible approach is to make the modeler fully responsible for modeling location-specific information about the artifact. Another option would be to automatically generate an extension of the model from a more high level specification. The latter approach has the advantage that the added information is consistent (e.g., a message can only be received after it has been sent). However, our subsequent treatment of location-aware artifacts does not depend on the way they have been obtained.

For the sake of automatically generating location information, we observe that the necessary extension to an artifact model can be reduced to applying a reasonably sized set of recurring patterns. In consequence, we suppose that it is possible to automatically derive the extension from a few general categories. In the next section, we make a preliminary proposal for such a categorization.

4 Categorization of location information

In this section, we propose a two-dimensional categorization of artifacts and discuss the consequences on the derivation of a location-aware extension of an artifact. The first dimension is concerned with the possible changes of ownership and remote visibility of the artifact. The second dimension deals with remote accessibility to actions.

In the first dimension, we distinguish *mobile*, *persistent*, and *transient* artifacts.

A mobile artifact may change its location over time. A typical example is a physical form that is exchanged between agents, for instance for collecting information or just signatures from different agents. The direct debit authorization discussed in previous sections is a particular instance of a mobile artifact. Messages caused by a mobile artifact typically correspond to a change of location of the artifact. This can be modeled using an additional data element that records the location which may be at a particular agent or "in transit" between two agents. Actions correspond to sending an artifact (move the location field from "at X" to "in transit from X to Y" and receiving an artifact (move the location field from "in transit from X to Y" to "at Y"). The location field may then be used for constraining remote access as discussed later in this section.

Persistent and transient artifacts are both immobile; that is, their location does not change over time. The difference between these two categories concerns the visibility of the artifact to other agents. An example for a persistent artifact could be a commonly known Web front-end such as a popular book-ordering platform. Access to actions on the artifact, including creation of an order (an actual instance of the artifact) can happen at any time from any agent. In contrast, consider a journal reviewing record as an example for a transient artifact. This artifact resides in the editorial office at all time — like the ordering artifact resides with the seller. However, the reviewer cannot access the artifact from the beginning as he or she simply does not know of its existence. Only after having been invited to review the paper, the reviewer can start to act on the artifact (including downloading the paper and filling in the fields in the recommendation form).

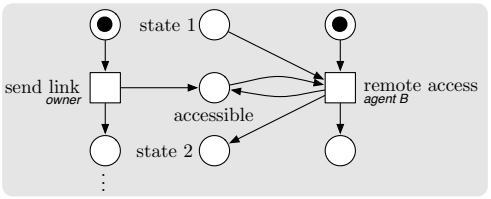


Fig. 3. Excerpt of a transient artifact whose *owner* needs to send a link prior to remote access of *agent B*.

In essence, the reviewer invitation contains a *link* to the artifact, possibly in the form of login information thus announcing the existence to the artifact. This link message makes the artifact remotely accessible. For a persistent artifact, no such information is required. At least, passing the link to the artifact to the remote (customer) agent, is typically not part of the interorganizational business process model for book selling.

Persistent artifacts basically do not require any location-specific extension (such as the **order** example). It is just necessary to be aware of the particular location for the purpose of distinguishing remote from local access to the artifact. For a transient model, we propose to add a place for each agent that is marked as soon as the artifact is visible to that agent. An action "send link" marks that place thus modeling the fact that the artifact can be accessed after having received the link (or login) information. Once a place is marked, the artifact can be accessed indefinitely by the respective agent. See Fig. 3 for an example.

The second dimension determines, whether and how an artifact can be accessed by remote agents. For a mobile artifact, an agent is remote if it is not currently owning the artifact. For a persistent or transient artifact, all agents are remote, except the one that currently possesses it. Remote accessibility may differ between actions, so we suggest to specify this information for each action separately.

We distinguish three options for remote accessibility of an action: *none*, *synchronous*, and *asynchronous*. For a real paper form, the standard option would be none. The form is not remote accessible. Performing an action requires physical presence of the artifact. An exception may be a situation where two agents are actually present in a single location such as in the case of a contract that is signed by a customer directly at a desk which does not require passing the contract from the clerk to the customer. Synchronous transfer is an obvious option for artifacts with interactive Web forms as front-end. An example for an asynchronously accessible artifact can be found in the once popular tool *Majordomo*[1] for managing electronic mailing lists. Participants could manipulate their recorded data (like subscription and unsubscription) by writing e-mails containing specific commands to a particular e-mail address.

Although there is a certain correlation between the dimensions, we can think of examples for all possible combinations of values for the two discussed dimensions. Even for a mobile artifact, asynchronous remote access may be reasonable. Think of a product that is about to be assembled where the delivery and mounting of a part from a supplier

[1] `http://www.greatcircle.com/majordomo/`

Table 1. Dimensions of location information with examples.

	no remote access	**synchronous remote access**	**asynchronous remote access**
mobile artifact	physical form	insurance claim with delegation	
persistent artifact	—	database	majordomo
transient artifact	—	online survey	review form

may be modeled as an asynchronous access to the artifact. Thus, there is a need to explicitly state the remote accessibility scheme for each action.

We do not claim that the above categorization (see Table 1 for an overview) is complete. However, we argue that this categorization provides enough evidence to support the claim that location-based information about artifacts can be systematically specified and then transformed into an extended artifact model. Potential refinements include, for example, restrictions concerning the set of agents to which the artifact may be passed or made visible. However, it is safe to assume for the remainder of this paper that a location-aware artifact model be given.

5 Choreography construction

With location-aware artifacts, we can model the evolution (life cycle) of distributed data objects. As these life cycles evolve independently, several problems may arise:

1. Each artifact may reach a local final state, but the respective global state might model an unreasonable combination of final states. For instance, a beer order together with a wine-loaded cargo is reachable in the running example.
2. Even worse, undesired situations such as deadlocks (nonfinal markings without successors) or livelocks (infinite runs without reachable final marking) might be reachable.
3. Furthermore, not every reachable behavior is actually permitted: Policies may additionally restrict the order of actions that reaches a final state. For instance, a shipper must not load the cargo before the ordered goods have been paid.

In the course of this section, we address these three problems as follows. With *goal states*, we restrict the set of all possible final states to a subset of desired global final states. This addresses the first problem. To avoid deadlocks and livelocks, the artifacts' actions need to be *controlled* by the environment, resulting in an interaction model (i.e., a choreography) which may serve as a contract between the agents. This interaction model provides the necessary coordination to deal with the second problem. Finally, we introduce *policies* to further refine the interdependencies between artifacts. This tackles the third problem. Figure 4 provides an overview.

5.1 Goal states and controller synthesis

To simplify subsequent definitions, we first unite the artifacts. The union of a set of artifacts is again an artifact.

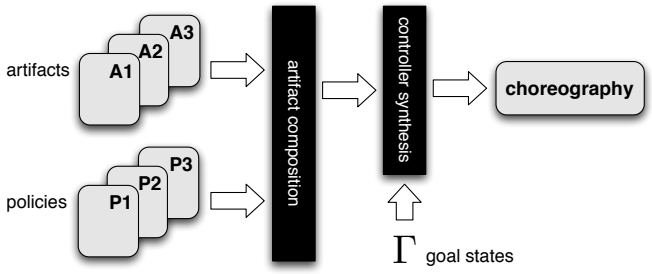


Fig. 4. Overview: Artifact composition and controller synthesis (Sect. 5.1) and policies (Sect. 5.2) yield a choreography.

Definition 3 (Artifact union). *Let $A_1, \ldots, A_n$ be artifacts with pairwise disjoint Petri nets $N_1, \ldots, N_n$. Define the* artifact union $\bigcup_{i=1}^{n} A_i = [N, \ell, \Omega]$ *to be the artifact consisting of*

- $N = [\bigcup_{i=1}^{n} P_i, \bigcup_{i=1}^{n} T_i, \bigcup_{i=1}^{n} F_i, m_{0_1} \oplus \cdots \oplus m_{0_n}]$,
- $\ell(t) = \ell_i(t)$ *iff* $t \in T_i$ *($i \in \{1, \ldots, n\}$), and*
- $\Omega = \{m_1 \oplus \cdots \oplus m_n \mid m_i \in \Omega_i \wedge 1 \leq i \leq n\}$

Thereby, $\oplus$ denotes the composition of markings: $(m_1 \oplus \cdots \oplus m_n)(p) = m_i(p)$ iff $p \in P_i$.

The previous definition is of rather technical nature. The only noteworthy property is that the set of final markings of the union consists of all combinations of final markings of the respective artifacts. We shall later restrict this set of final markings to a subset of *goal states*.

The next definition captures the interplay between two artifacts A_1 and A_2 and uses their interfaces (i.e., the labels associated to artifact transitions) to synchronize artifacts. In the resulting *composition*, each pair of transitions t_1 and t_2 of artifact A_1 and A_2, respectively, with the same label (i.e., $\ell_1(t_1) = \ell_2(t_2)$) is replaced by a new transition $[t_1, t_2]$ which models synchronous firing of t_1 and t_2. Consequently, the composition of two artifacts restricts their behavior by synchronization.

Definition 4 (Artifact composition). *Let A_1 and A_2 be artifacts. Define their* shared labels *as $S = \{l \mid \exists t_1 \in T_1, \exists t_2 \in T_2 : \ell(t_1) = \ell(t_2) = l\}$. The* composition *of A_1 and A_2 is the artifact $A_1 \oplus A_2 = [N, \ell, \Omega]$ consisting of:*

- $N = [P, T, F, m_{0_1} \oplus m_{0_2}]$ *with*
  - $P = P_1 \cup P_2$*;*
  - $T = \big(T_1 \cup T_2 \cup \{[t_1, t_2] \in T_1 \times T_2 \mid \ell(t_1) = \ell(t_2)\}\big) \setminus \big(\{t \in T_1 \mid \ell_1(t) \in S\} \cup \{t \in T_2 \mid \ell_2(t) \in S\}\big)$,
  - $F = ((F_1 \cup F_2) \cap ((P \times T) \cup (T \times P))) \cup \{[[t_1, t_2], p] \mid [t_1, p] \in F_1 \vee [t_2, p] \in F_2\} \cup \{[p, [t_1, t_2]] \mid [p, t_1] \in F_1 \vee [p, t_2] \in F_2\}$,
- *for all $t \in T \cap T_1$: $\ell(t) = \ell_1(t)$, for all $t \in T \cap T_2$: $\ell(t) = \ell_2(t)$, and for all $[t_1, t_2] \in T \cap (T_1 \times T_2)$: $\ell([t_1, t_2]) = \ell_1(t_1)$, and*

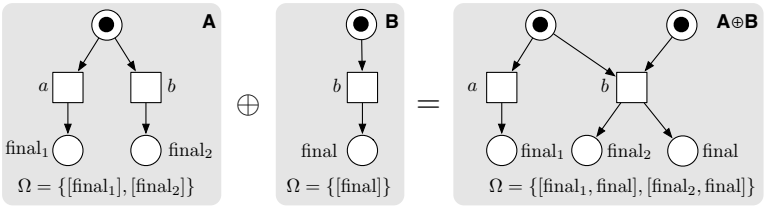


Fig. 5. Example for the composition of two artifacts.

– $\Omega = \{m_1 \oplus m_2 \mid m_1 \in \Omega_1 \wedge m_2 \in \Omega_2\}$.

The composition $A_1 \oplus A_2$ *is* complete *if for all* $t_1 \in T_1$ *and* $t_2 \in T_2$ *holds:* $\ell_1(t_1) \in S$ *and* $\ell_2(t_2) \in S$.

Figure 5 depicts an example for the composition of two artifacts. Final markings of the composition are built just like in the union. We call a composition *complete* if for each transition in one artifact exists a transition in the other artifact that carries the same label. Intuitively, a complete composition does not contain "unsynchronized" transitions. To avoid undesired behavior, a complete composition plays an important role.

Given an artifact A and a set $\Gamma \subseteq \Omega$ of *goal states* of A, we call another artifact A' a *controller* for A iff (1) their composition $A \oplus A'$ is complete and (2) for each reachable markings of the composition, a marking $m \oplus m'$ is reachable such that $m \in \Gamma$ and m' is a final marking of A'. Intuitively, this controller synchronizes with A such that a goal state $m \in \Gamma$ of A always remains reachable.

The existence of controllers (also called *controllability* [24]) is a fundamental correctness criterion for communicating systems such as services. It can be decided constructively [24]: If a controller for an artifact exists, it can be constructed automatically.

With the concept of controller synthesis, we are now able to reason about artifacts. Given a set of artifacts and a set of goal states, we can synthesize a controller which rules out any behavior that makes the goal states unreachable. At the same time, the controller provides a global model which specifies the order in which the agents may perform actions on the artifacts. Furthermore, we can derive communication actions from the labels that were introduced in the location-aware versions.

For the three artifacts of the running example and the set of goal states

$$\Gamma = \{[\text{used}, \textbf{debit}@\text{shipper}, \text{paid}, \text{beerOrdered}, \text{beerLoaded}, \textbf{cargo}@\text{buyer}],$$
$$[\text{used}, \textbf{debit}@\text{shipper}, \text{paid}, \text{wineOrdered}, \text{wineLoaded}, \textbf{cargo}@\text{buyer}]\}$$

expressing successful purchase of beer or wine, respectively, the synthesized controller has 1120 states and is too large to be shown here. Although free of deadlocks and livelocks, it still contains undesired behavior which we rule out with policies in the next subsection.

Table 2. Required policies to rule out unintended behavior

policy	description	artifacts
P1	only load cargo after buyer has paid	**order**, **cargo**
P2	only pay when filled debit form is at shipper	**order**, **debit**
P3	do not send unloaded cargo to buyer	**cargo**
P4	only send debit form if it is filled and at the buyer	**debit**

5.2 Policies

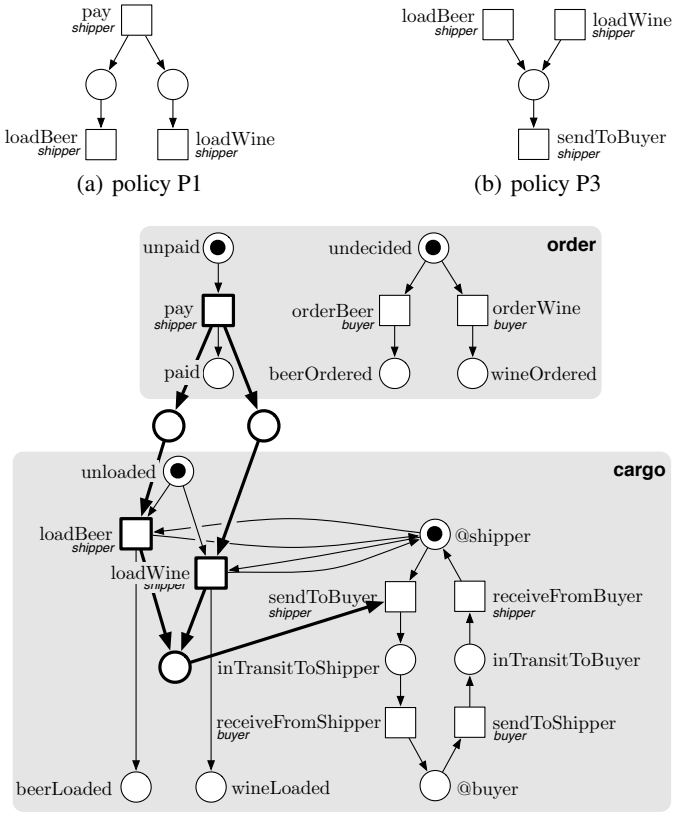
Artifact-centric approaches follow a declarative modeling style. The same holds for artifact-centric choreographies: instead of explicitly modeling global state changes, we only modeled the local object life cycle of every artifact. Consequently, the order of actions in the generated choreography is only constrained to avoid deadlocks and livelocks with respect to goal states. As a downside of this approach, a lot of unreasonable behavior is exposed. For instance, sending of unloaded cargo or even sending without prior payment is possible.

To rule out this undesired behavior, we employ *policies* (also called *behavioral constraints* [18]). In the setting of this paper, we also model policies with artifacts; that is, labeled Petri nets with a set of final markings. These artifacts have no counterpart in reality and are only used to model dependencies between actions of different artifacts. The application of policies then boils down to the composition of the artifacts with these policies. In principal, goal states can be expressed by policies as well. We still decided to split these concepts, because the former express liveness properties whereas the latter express safety properties.

For the running example, we used four policies, listed in Table 2. To avoid, for instance, loading the cargo without payment (P1), the **order** and **cargo** artifacts need to be constrained. Specifically, the **cargo**'s transitions with label "loadBeer" and "loadWine" must not fire before **order**'s transition "pay" has fired. Such a dependency can be straightforwardly expressed by Petri nets, for instance by the constraint net in Fig. 6(a). Figure 6(b) depicts the similar constraint net for policy P3. Their composition with the **order** and **cargo** artifacts is shown in Fig. 6(c).

By applying more and more policies to the artifacts, we exclude more and more unintended behavior. This effect can be observed in the number of states of the choreography. Without any constraints, the choreography has 1120 states. After the application of policy P1, this number is reduced to 736. With more policies, this number is quickly reduced to 216 (P1, P2), 60 (P1, P2, P3), and finally 30 states with all four policies.

Figure 7 depicts the Petri net representing this final choreography. Each transition is labeled with an action (what is done), an artifact (which data are accessed), and an agent (who performs the action). This Petri net model exposes the concurrency between the handling of the direct debit authorization and the buyer's order choice. This choreography has been calculated by Wendy [19] as an automaton model which then was transformed into a Petri net model using the tool Genet [11]. With a preprocessing tool to compose artifacts and policies, Wendy as controller synthesis tool, and Genet as Petri net transformation, we have a continuous tool chain for location-aware artifact-centric



(a) policy P1 (b) policy P3

(c) policy P1 and P3 applied to artifacts **order** and **cargo**.

Fig. 6. Application of policies to artifacts.

choreographies available. Although the running example is rather trivial, case studies with Wendy [19] show that it is able to cope with input models with millions of states.

6 Related work

There exists a variety of approaches dealing with artifact-centric modeling of business processes.

Workflow construction Küster et al. [17] study the interplay between control flow models and object life cycles. Beside an extended soundness notion which also respects object life cycles, the authors present an algorithm to automatically derive a sound process model from given object life cycles. Compared to our approach, this workflow model is not aware of artifact locations and hence plays an orchestrating role.

Object life cycle inheritance Aalst and Basten [2,3] model object life cycles as labeled Petri nets and investigate the relationships between different versions of object life cycles.

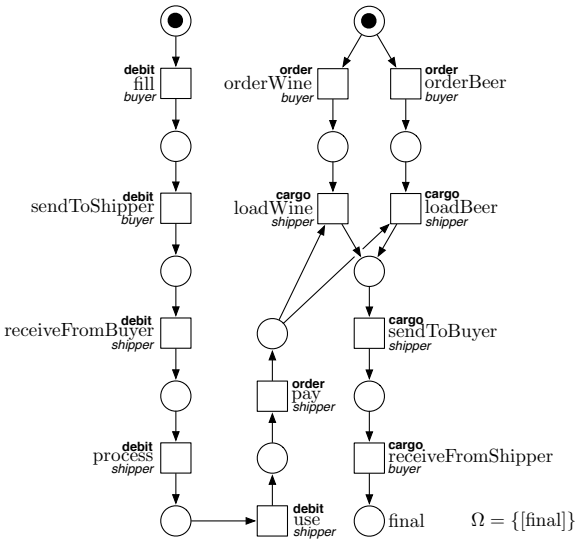


Fig. 7. Resulting choreography between *buyer* and *shipper*.

These *inheritance* notions may complement our work and can, for instance, be used in our context to compare artifacts and choreographies. Such an application to contracts is described by Aalst et al. [4].

Artifacts and service orientation The idea of encapsulating functionality as services also influenced artifact-centric approaches. Several authors investigated how services can be used to manipulate the state of artifacts. Keeping a declarative modeling style, services are described by preconditions and postconditions formulated in different logics.

Bhattacharya et al. [8] study several questions related to artifacts and services, including reachability of goal states, absence of deadlocks, and redundancy of data. Their employed logics is similar to OWL-S. Similar settings are investigated by Calvanese et al. [10] and Fritz et al. [13] for first order logics. Gerede and Su [14] language based on CTL to specify artifact behaviors in artifact-centric process models. Each paper provides complexity and decidability results for the respective problems.

These approaches share the idea of using service calls to manipulate artifacts. The artifact itself, however, is assumed to be immobile, resulting in orchestrating workflows rather than our choreography-like setting.

Artifact hosting Hull et al. [15] introduce *artifact-centric hubs* as central infrastructure hosting data that can be read and written by participants. The authors motivate that, compared to autonomous settings such as choreographies, the centralized storage of data has the advantage of providing a conceptual rendezvous point to exchange status information. This centralized approach can be mimicked by our location-aware approach by remotely accessible immobile artifacts.

Proclets Aalst et al. [1,5] introduce *proclets* to specify business processes in which object life cycles can be modeled in different levels of granularity and cardinality.

Consequently, proclets are well-suited to deal with settings in which several instances of data objects are involved. Being introduced as workflow models, proclets have no concept of locations.

To the best of our knowledge, this is the first approach to explicitly add information on agents and locations to artifact models. As a result, we can naturally reason about sending and receiving of artifacts resulting in a choreography of agents. Consequently, our approach is a first step toward artifact-based *interorganizational* business processes.

7 Conclusion

We extended the idea of artifact-centric process design to a choreography setting where artifacts are not necessarily gathered in artifact hubs. We observed that in such a setting the actual location of the artifact has a significant impact on executability of actions and on the message flow that is required to transform artifacts into desired goal states. We propose to enhance artifacts with explicit information on location and its impact on remote access to actions. This information can be modeled manually or derived systematically from a high level description. We suggest a principal two-dimensional categorization into mobile, persistent, and transient artifacts on one hand, and no, synchronous, or asynchronous remote access to actions on the other as an initial proposal for a high level description. From location-aware artifacts and goal states for the artifacts, we can derive a global interaction model that may serve as a contract between the involved agents. The interaction model can be derived in such a way that it respects specified policies of the involved agents. The whole approach relies on only one simple formalism. Petri nets express the functional part of an artifacts, location information, as well as policies. This way, it is possible to employ existing tools for the automated construction of a choreography and the invocation of policies. These tools have already proven their capability to cope with nontrivial problem instances.

In our modeling approach to artifacts, we did not include mechanisms for creating new artifact instances. If the overall number of artifacts in the system is bounded, this is not a serious problem since the creation of a new artifact instance can be modeled by a transition from a state "not existing" to the actual initial state of the artifact. This approach does not work in the case of an unbounded number of artifacts. Similar problems are known in the area of verification of parameterized programs where parts of the program may spawn a finite but not a priori bounded number of threads which run identical programs. There exist ways to finitely model such systems and several verification problems turn out to be decidable [12,6]. Future research is required to find out whether the methodology used there extends to the problems and solutions proposed in this paper.

Another interesting issue is the further transformation of the choreography derived in this paper into local processes for the agents. We see two potential directions which need to be further explored. First, we could exploit existing research on *accordance* (e.g., [7,9,23]). In [4], we showed that it is possible for each agent to replace its part of a contract by an accordant private process. Relying on the accordance criterion, soundness of the original interaction model is inherited by the collaboration of private processes.

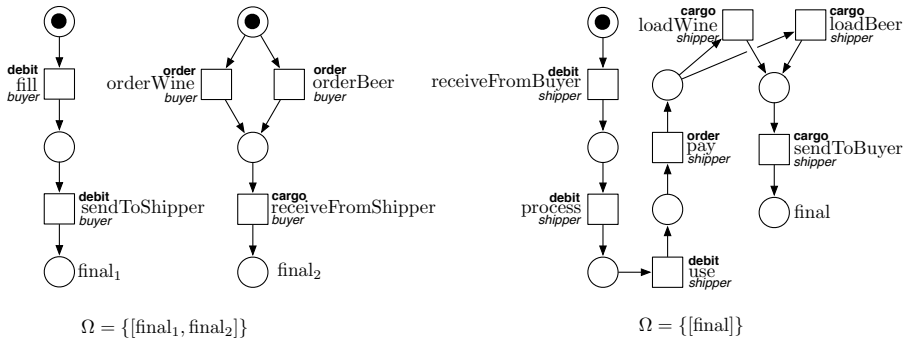


Fig. 8. Local models for the *buyer* (left) and the *shipper* (right).

The approach requires a suitable decision procedure for checking accordance [23,22] or powerful transformation rules which preserve accordance [16]. Both appear to be more advanced for establishing deadlock freedom than for livelock freedom, so more progress needs to be made there.

A second opportunity for deriving local processes is to use the *realizability* approach proposed in [20]. There, local processes are constructed from a choreography for the sake of proving that the choreography can be implemented (realized). To this end, the choreography is transformed into a service where the local processes are computed as correctly interacting partners, cf. Fig. 8. Adding results from [24] to this approach, we can even compute a finite representation of a *set* of processes for each agent such that each combination of one process per agent yields a correct set of realizing partners for the choreography. The concept was called *autonomous controllability* in [24]. In the artifact setting, such a finite representation of a set of processes could be used to derive, at least to some degree, a local process that not only respects the artifacts and policies known to all involved agents, but also artifacts and policies that are hidden from the other agents. Again, past research focused on deadlock freedom, so further work is required to make that technology available in the context of this paper. Nevertheless, the discussion suggests that the chosen approach connects artifact centric choreographies to promising methods for further tool support.

We believe that we can further exploit ideas for bridging the gap between the global interaction model and the local processes of the agents. It is also worth to explore ideas known from program verification for the purpose of supporting unbounded creation of artifact instances.

References

1. Aalst, W.M.P.v.d., Barthelmess, P., Ellis, C.A., Wainer, J.: Proclets: A framework for lightweight interacting workflow processes. Int. J. Cooperative Inf. Syst. 10(4), 443–481 (2001)
2. Aalst, W.M.P.v.d., Basten, T.: Life-cycle inheritance: A Petri-net-based approach. In: PETRI NETS 1997. pp. 62–81. LNCS 1248 (1997)

3. Aalst, W.M.P.v.d., Basten, T.: Identifying commonalities and differences in object life cycles using behavioral inheritance. In: PETRI NETS 2001. pp. 32–52. LNCS 2075 (2001)
4. Aalst, W.M.P.v.d., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: Multiparty contracts: Agreeing and implementing interorganizational processes. Comput. J. 53(1), 90–106 (Jan 2010)
5. Aalst, W.M.P.v.d., Mans, R.S., Russell, N.C.: Workflow support using proclets: Divide, interact, and conquer. IEEE Data Eng. Bull. 32(3), 16–22 (2009)
6. Ball, T., Chaki, S., Rajamani, S.K.: Parameterized verification of multithreaded software libraries. In: TACAS 2001. pp. 158–173. LNCS 2031, Springer (2001)
7. Benatallah, B., Casati, F., Toumani, F.: Representing, analysing and managing Web service protocols. Data Knowl. Eng. 58(3), 327–357 (2006)
8. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: BPM 2007. pp. 288–304. LNCS 4714, Springer (2007)
9. Bravetti, M., Zavattaro, G.: Contract based multi-party service composition. In: FSEN 2007. pp. 207–222. LNCS 4767, Springer (2007)
10. Calvanese, D., Giacomo, G.D., Hull, R., Su, J.: Artifact-centric workflow dominance. In: ICSOC/ServiceWave 2009. pp. 130–143. LNCS 5900, Springer (2009)
11. Carmona, J., Cortadella, J., Kishinevsky, M.: Genet: A tool for the synthesis and mining of petri nets. In: ACSD 2009. pp. 181–185. IEEE Computer Societey (2009)
12. Emerson, E.A., Kahlon, V.: Reducing model checking of the many to the few. In: CADE 2000. pp. 236–254. LNCS 1831, Springer (2000)
13. Fritz, C., Hull, R., Su, J.: Automatic construction of simple artifact-based business processes. In: ICDT 2009. pp. 225–238. ACM International Conference Proceeding Series 361, ACM (2009)
14. Gerede, C.E., Su, J.: Specification and verfication of artifact behaviors in business process models. In: ICSOC 2007. pp. 181–192. LNCS 4749, Springer (2007)
15. Hull, R., Narendra, N.C., Nigam, A.: Facilitating workflow interoperation using artifact-centric hubs. In: ICSOC/ServiceWave 2009. pp. 1–18 (2009)
16. König, D., Lohmann, N., Moser, S., Stahl, C., Wolf, K.: Extending the compatibility notion for abstract WS-BPEL processes. In: WWW 2008. pp. 785–794. ACM (Apr 2008)
17. Küster, J.M., Ryndina, K., Gall, H.: Generation of business process models for object life cycle compliance. In: BPM 2007. pp. 165–181. LNCS 4714, Springer (2007)
18. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral constraints for services. In: BPM 2007. pp. 271–287. LNCS 4714, Springer (Sep 2007)
19. Lohmann, N., Weinberg, D.: Wendy: A tool to synthesize partners for services. In: PETRI NETS 2010. pp. 297–307. LNCS 6128, Springer (2010), tool available at `http://service-technology.org/wendy`.
20. Lohmann, N., Wolf, K.: Realizability is controllability. In: WS-FM 2009. pp. 110–127. LNCS 6194, Springer (2010)
21. Nigam, A., Caswell, N.S.: Business artifacts: An approach to operational specification. IBM Systems Journal 42(3) (2003)
22. Stahl, C.: Service Substitution - A Behavioral Approach Based on Petri Nets. Ph.D. thesis, Humboldt-Universität zu Berlin, Mathematisch-Naturwissenschaftliche Fakultät II; Eindhoven University of Technology (2009)
23. Stahl, C., Massuthe, P., Bretschneider, J.: Deciding substitutability of services with operating guidelines. LNCS T. Petri Nets and Other Models of Concurrency 2(5460), 172–191 (2009)
24. Wolf, K.: Does my service have partners? LNCS T. Petri Nets and Other Models of Concurrency 5460(2), 152–171 (2009)