Chapter 1
From artifacts to activities

Niels Lohmann and Karsten Wolf

Abstract We consider services as units in interorganizational business processes. Following trends in the business process management community, we switch from an activity-centric description of processes to artifact-centric descriptions. In the interorganizational setting, unique problems arise. For instance, an artifact hub that is crucial for present-day enactment of artifact-centric processes, can hardly be shared between different organizations since the stored information may be subject to trade secrets. We propose a solution that involves the translation of an artifact-centric model into an activity-centric model. In this course, we consider artifacts as entities that may be sent around between organizations. The location of an artifact may imply access restrictions for one or the other organization. We propose both a formal model and algorithms to show the effectiveness of our approach.

1.1 Introduction

Different communities are concerned with web services. Consequently, there exist several different views on the topic. Some emphasize technical issues while others focus on business aspects. In the area of business process management (BPM), a web service is often understood as a substructure in an interorganizational business process. The idea is that the service describes one party's share in the overall process. In this context, a service is typically a (local) business process where some tasks are communication activities with other parties. Communication may be arbitrarily complex thus distinguishing these services from the frequently advocated simple invoke/response scheme. Furthermore, these services have a meaningful internal state determined by the respective state of the local business process. Hence, techniques such as reasoning about preconditions and postconditions are less appropriate than for simple invoke/response services. In addition, unique problems pop up such as the deadlock freedom of the communication between organizations.

Niels Lohmann · Karsten Wolf
University of Rostock, e-mail: niels.lohmann|karsten.wolf@uni-rostock.de

A typical business process model consists of activities that are arranged using control flow primitives such as sequential or parallel execution, exclusive or inclusive branches, and loops. We shall refer to such a process representation as an *activity-centric* process. Currently, this is the dominating way of describing business processes and most workflow engines and modeling notations rely on activity-centric process models. However, alternative representations have been discussed as well. One of these alternatives are so-called *artifact-centric* processes. Promoters claim that artifact-centric models are better comprehendible by business analysts (which typically have little background in computer science). An artifact-centric process explicitly represents business-relevant artifacts such as documents, database states, etc. together with their distinguished life cycle and certain goals (or milestones). The actual activities remain implicit. An example for an artifact would be an application form which may have the simple life cycle not filled/filled but not signed/signed. If the goal is to get a signed form, we can derive the sequence of activities *fill/sign*. In most proposals for enacting artifact-centric processes, the derivation of activities is left to an AI planner that runs on the data of the actual process instance. To this end, all process relevant data need to be stored in a single database that is called *artifact hub*.

We consider this approach as quite problematic for several reasons. First, problems of unsoundness of the overall approach are detected only at run time which may cause severe threats for real business relations. Second, AI planning is a computationally challenging problem, so the execution of a planner at run time may cause unacceptable delays in process execution. Third, and most relevant in our interorganizational setting, trade secrets between the involved companies may inhibit a central artifact hub. The contribution of this chapter is to address these problems. First, we propose to automatically translate an artifact-centric process into an activity-centric process. This way, we keep the advantages of an artifact-centric view to the business analyst while being able to map execution to existing and mature workflow engines. Time expenses are shifted from run time (as for the AI planner) to preparation time (for the translation) which is less time-critical. Moreover, translation inherently includes a check for soundness of the derived model. This part of our contribution is presented in Sect. 1.4. Then, in Sect. 1.5, we propose techniques that allow us to abandon the artifact hub. In our approach, an artifact may be a mobile entity; that is, one that may be sent from one organization to another. This way, we can model artifacts that are invisible or inaccessible to some organization for certain points in time. We can further directly derive the necessity to perform communication activities from the artifact model. As an example, consider the above-mentioned application form. If filling and signing the form is performed by different business units, we observe that both filling and signing is only possible if the form is physically present in the respective unit. Under the assumption that the empty form is initially available for the filling unit, we would be able to come up with the activity-centric model *fill/send to other unit* for the one business unit and *receive filled form/sign* for the other. Consequently, we first propose a model for mobility of artifacts, cf. Sect. 1.5.1. Then, we augment this model with other information that is useful for deriving faithful activity-centric models from a given artifact-centric model. An example would be the enforcement of compliance rules, cf. Sect. 1.5.2. Finally, we discuss related work (Sect. 1.6) and conclude (Sect. 1.7).

All together, we thus outline an approach for an artifact-centric service collaboration. This presentation is based on previous papers of the authors [43, 37, 38].

1.2 Running example: insurance claim handling

We use a simple insurance claim handling process (based on [52]) as running example for this chapter. In this process, a customer submits a claim to an insurer who then prepares a fraud detection check offered by an external service. Based on the result of this check, the claim is either (1) assessed and the settlement estimated, (2) detected fraudulent and reported, or (3) deemed incomplete. In the last case, further information are requested from the customer before the claim is resubmitted to the fraud detection service. In this situation, the customer can alternatively decide to withdraw the claim. On successful assessment, a settlement case is processed by a financial clerk. The claim is settlement paid in several rates or all at once. A single complete payment further requires an authorization of the controlling officer. When the settlement is finally paid, the claim is archived.

1.3 A formal model for artifacts

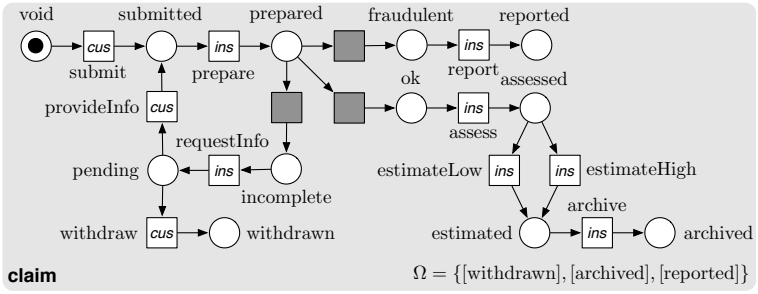
Being concerned with an interorganizational setting, we assume that there is a set $\mathcal{A}$ of *agents* (or roles, organizations, etc.). In our approach, agents are principals in a role-based access control for artifacts.

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.

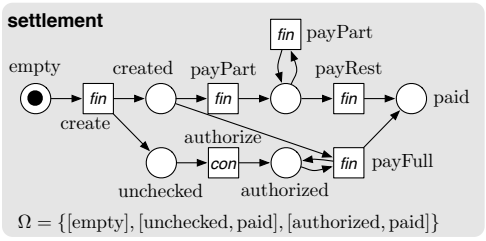
Definition 1 (Petri net). A *Petri net* $N = [P, T, F, m_0]$ consists of two finite and disjoint sets P (*places*) and T (*transitions*), a *flow relation* $F \subseteq (P \times T) \cup (T \times P)$, and an *initial marking* m_0. A marking $m : P \to \mathbb{N}$ represents a state of the Petri net 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.

Throughout this chapter fix a set $\mathcal{L} = \mathcal{L}_c \cup \mathcal{L}_u$ of *action labels*. This set is partitioned into a set $\mathcal{L}_c$ of *controllable actions* that are executed by agents and a set $\mathcal{L}_u$ of *uncontrollable actions* that are not controllable by any agent, but are under the influence of the environment. Such uncontrollable actions are suitable to model choices that are external to the business process model, such as the outcome of a service call (e.g., to a fraud detection agency) or just choices whose decision process



(a) **claim** artifact



(b) **settlement** artifact

Fig. 1.1 Running example process

is not explicitly modeled at this level of abstraction. We further define a mapping $c : \mathcal{L} \to \mathcal{A}$ representing the *access control*. This access control can be canonically extended [38] to sets of agents (i.e., *roles*), yielding a sophisticated role-based access control.

Running example (cont.) Figure 1.1 depicts the claim and the settlement artifacts. Each transition is labeled by the agent that executes it (*ins*urer, *cus*tomer, *con*troller, and *fin*ancial clerk) or is shaded gray in case of uncontrollable actions.

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 markings of the **claim** artifact include the markings [withdrawn], [reported], and [archived] modeling the different outcomes of the claim handling. The **settlement** artifact also has three final markings: [empty] (no settlement has been created), [unchecked, paid] (unchecked payment), and [authorized, paid] (authorized payment).

1.4 Executing artifact-centric business processes

In this section, we present the first part of our approach: the translation of an artifact-centric model into an activity-centric one. For the moment, we ignore the interorganizational aspects of our setting. These aspects are added in the next section.

At first glance, execution of an artifact-centric process amounts to a sequence of activities that transforms all artifacts into their respective goal states. A second view, however, shows that additional aspects need to be taken into consideration.

1. Even if every artifact reaches its local final state, the respective global state might still model an unreasonable and undesired situation. For instance, a beer order together with a wine-loaded cargo is reachable in the running example but certainly unwanted.
2. Even if a global sequence of activities is reasonable with respect to every artifact in isolation, it may introduce problems such as deadlocks (nonfinal markings without successors) or livelocks (infinite runs without reachable final marking).
3. Even if a sequence of activities is formally correct from the control flow perspective, it may not be meaningful from a semantic perspective. For instance, a shipper must not load the cargo before the ordered goods have been paid although that sequence would perfectly transform all artifacts into their final states while avoided deadlocks or lovelocks.

In the course of this section, we address these three problems as follows. With a specification of *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. Later, in Sect. 1.5 we further discuss how compliance rules can be integrated into this approach. Figure 1.2 provides an overview.

1.4.1 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.

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 $p \in P_i$.

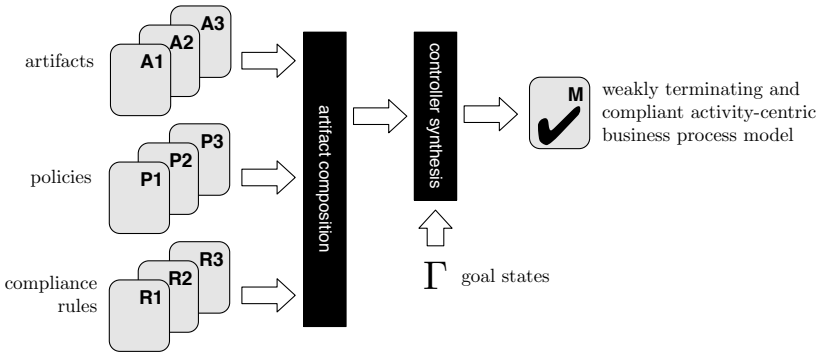


Fig. 1.2 Overview: Artifact composition and controller synthesis (Sect. 1.4.1) derive a choreography from artifacts, policies (Sect. 1.4.2), and compliance rules (Sect. 1.5).

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 $S = \mathcal{L}_c \setminus \{l \mid \exists t_1 \in T_1, \exists t_2 \in T_2 : \ell(t_1) = \ell(t_2) = l\}$. 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
- $\Omega = \{m_1 \oplus m_2 \mid m_1 \in \Omega_1 \wedge m_2 \in \Omega_2\}$.

The composition $A_1 \oplus A_2$ is *complete* if for all $t \in T_i$ holds: if $\ell_i(t) \notin S$, then $\ell_i(t) \in \mathcal{L}_u$ ($i \in \{1, 2\}$).

Figure 1.3 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

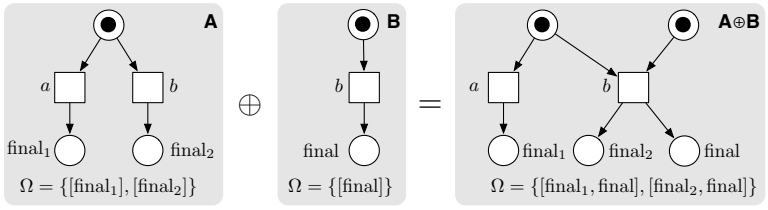


Fig. 1.3 Example for the composition of two artifacts.

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* [57]) is a fundamental correctness criterion for communicating systems such as services. It can be decided constructively [57]: 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 actions are performed on the artifacts.

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

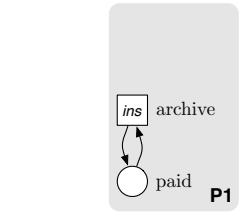
$$\Gamma = \{[\text{withdrawn}, \text{empty}], [\text{reported}, \text{empty}], \\ [\text{archived}, \text{unchecked}, \text{paid}], [\text{archived}, \text{authorized}, \text{paid}]\}$$

expressing withdrawn and reported claims as well as unchecked and authorized payments. Although free of deadlocks and livelocks, it still contains undesired behavior which we rule out with policies in the next subsection.

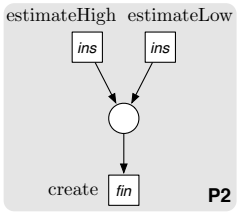
1.4.2 Policies

Artifact-centric approaches follow a declarative modeling style. 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, paying a settlement before the claim is assessed would be possible.

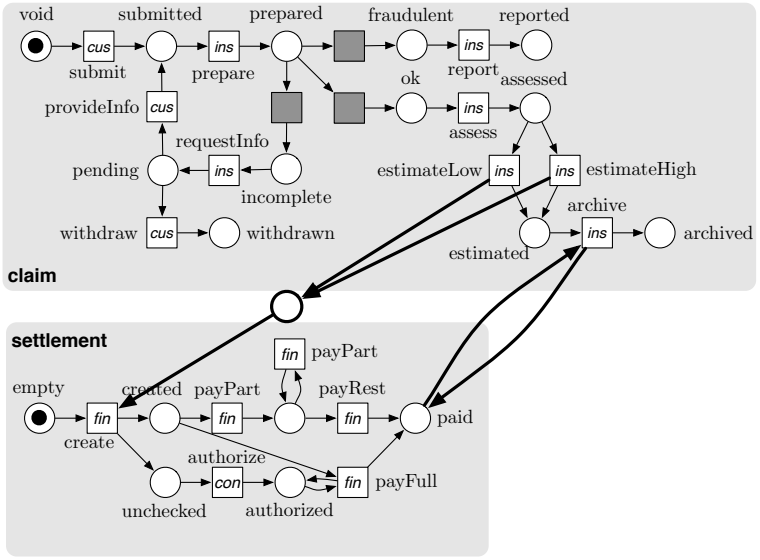
To rule out this undesired behavior, we employ *policies* (also called *behavioral constraints* [40]). For keeping notations slim, we also model policies with artifacts; that is, labeled Petri nets with a set of final markings. These artifacts have no physical



(a) policy P1



(b) policy P2



(c) policy P2 and P3 applied to artifacts **claim** and **settlement**.

Fig. 1.4 Application of policies to artifacts.

counterpart 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 conceptionally express liveness properties whereas the latter express safety properties.

For the running example, we use two policies:

P1	*The claim may be archived only if the settlement is paid.*
P2	*A settlement may only be created after the claim has been estimated.*

Figure 4(a) and 4(b) depict the artifacts for the policies **P1** and **P2**. The union with the artifacts **claim** and **settlement** is depicted in Fig. 4(c) where the policies are highlighted with bold strokes. By applying more and more policies to the artifacts,

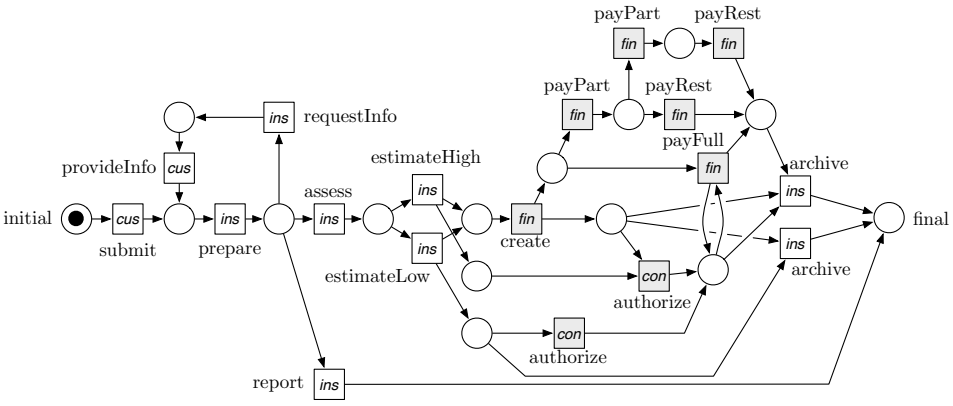


Fig. 1.5 Resulting choreography between agents *customer*, *insurer*, *controller* and *financial clerk*.

we add more dependencies between artifacts and exclude more and more unintended behavior.

Figure 1.5 depicts the Petri net representing the final activity-centric model. Each transition is labeled with an action (what is done), an artifact (which data are accessed; the actions on the **settlement** are shaded grey), and an agent (who performs the action). This choreography has been calculated by Wendy [42] as an automaton model which then was transformed into a Petri net model using the tool Genet [15]. 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 the translation from artifact-centric to activity-centric process models available. This tool chain offers an alternative to running an artifact hub at runtime. Especially, the approach is able to detect potential problems in an early phase of workflow engineering. Although the running example is rather trivial, case studies with Wendy [42] show that it is able to cope with input models with millions of states.

Note that the model is generated to realize the maximal behavior while guaranteeing correctness with respect to goal states, policies, and livelock freedom. Its main purpose is to be executed on standard workflow engines. As a result, the model is not necessarily human-readable. However, the synthesis tool Wendy also allows to generate smaller models with more restricted behavior. These models may be much easier to comprehend. Nevertheless, only the artifact models are meant to be understood by the modeler.

For models that cannot be controlled, we further introduced a diagnosis algorithm [36] that calculates a counterexample that makes the reasons for the impossibility to control the model toward deadlock and livelock freedom explicit. More details can be found in [38].

1.5 Extensions

Now we turn to the main contribution of this paper and approach the interorganizational aspects of artifact-centric processes. These aspects include the following issues:

- An artifact hub is even less suitable for enacting an interorganizational artifact-centric process;
- Artifacts may circulate between organizations, and their location may influence whether or not activities are enabled;
- In interorganizational collaborations, issues like compliance to legal requirements have increased importance.

While we addressed the first issue already in the previous section, we propose an explicit modeling of location as a solution for the second problem. That is, we extend the artifact model with states that represent the possible locations of the artifacts, and with activities that represent the transfer from one location to another one. The additional parts of the model are then linked to other activities for modeling location based restrictions for their activation. In the end, the extended model can be treated in the same way as proposed in the previous section. The resulting activity-centric model is a global business process model that provides a global view. It can serve as a contract (or a choreography, or a protocol) for the overall activity behavior. In the same style, we propose model extensions for coping with compliance requirements.

1.5.1 Interorganizational business processes

1.5.1.1 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 interorganizational interaction 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 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 bound to 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 current 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.

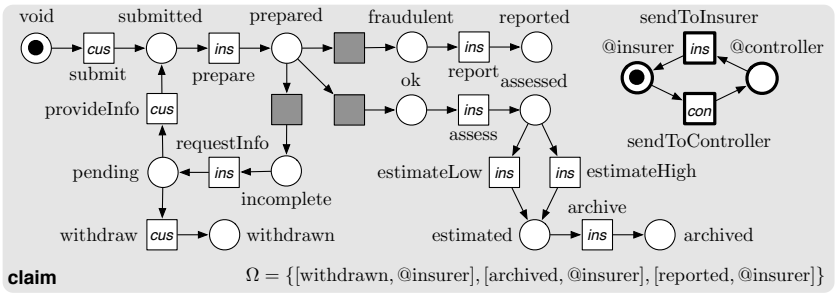


Fig. 1.6 Location-aware **claim** artifact. Added nodes are depicted with bold lines

Running example (cont.) Figure 1.6 depicts an location-aware extension of the **claim** artifact. Two additional places ("@insurer" and "@controller") model the *locaction* of the insurance claim file. In our example, we assume that after submitting the claim, a physical file is created by the insurer which can be sent to the controlling officer by executing the respective action "send to controller". We further assume that the **settlement** artifact is a data base entry that can be remotely accessed by the insurer, the financial clerk, and the controlling officer. Hence, we do not need to extend it with location information and keep the model in Fig. 1(b) as is.

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 by construction (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 subsection, we make a preliminary proposal for such a categorization.

1.5.1.2 Categorization of location information

In this subsection, 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

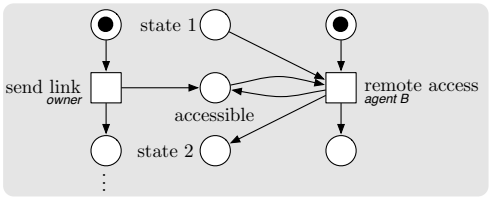


Fig. 1.7 Excerpt of a transient artifact whose *owner* needs to send a link prior to remote access of *agent B*.

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). 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 **settlement** 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. 1.7 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

Table 1.1 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

that 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 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 1.1 for an overview) is complete. However, discussions subsequent to the presentation of the original article [43] did not reveal any further categories. Further investigations of more involved scenarios such as Enterprise Integration Patterns [27] or Service Interaction Patterns [6] are subject of future work. It is this safe to assume for the remainder of this paper that a location-aware artifact model be given.

Running example (cont.) Locations can be used to refine the policies of our insurance process:

P1 *The claim may be archived only if it resides at the insurer and the settlement is paid.*
P2 *A settlement may only be created after the claim has been estimated.*
P3 *To authorize the complete payment of the settlement, the claim artifact must be at hand to the controlling officer.*

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

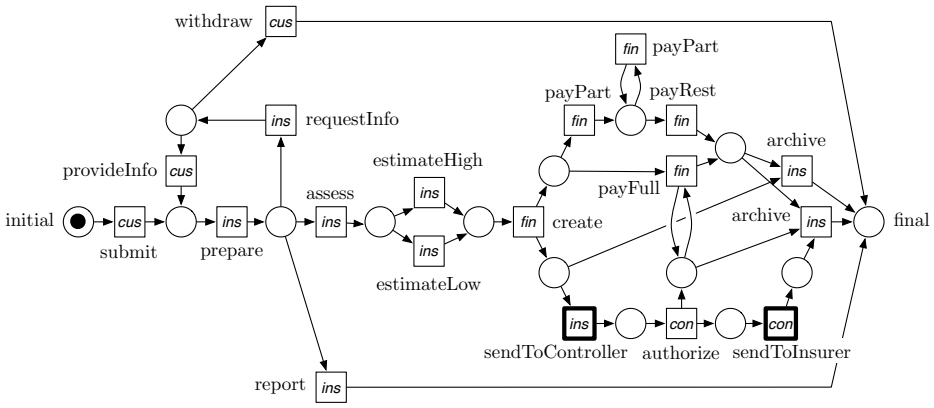


Fig. 1.8 Location-aware choreography with highlighted message exchange that satisfies policies P1–P5

P4 *The claim artifact may only be sent to the controller if it has been estimated and the settlement has not been checked.*
P5 *The claim artifact may only be sent back to the insurer if the settlement has been authorized.*

Figure 1.8 depicts the resulting location-aware choreography. Note the highlighted transitions modeling the required message exchange between the insurer and the controlling officer to satisfy policy P3–P5.

1.5.2 Compliance rules

Compliance rules are often *declarative* and describe *what* should be achieved rather than *how* to achieve it. Temporal logics such as CTL [10], LTL [50] or PLTL [51] are common ways to formalize such declarative rules. To make these logics approachable for nonexperts, also graphical notations have been proposed [5, 7]. Given such rules, compliance of a business process model can be verified using model checking techniques [16]. These checks can be classified as *compliance by detection*, also called after the fact or retrospective checking [53]. Their main goal is to provide a rigorous proof of compliance. In case of noncompliance, diagnosis information may help to fix the business process toward compliance. This step can be very complicated, because the rules may affect various parts and agents of the business process (e. g., financial staff and the press team). Furthermore, the declarative nature of the rules does not provide recipes on how to fix the business process. To meet the previous example rule, an action “send information to press team” needs to be added to the process and must be executed at most two weeks after the execution of an action “sign financial report”. Compliance can be eventually reached after iteratively adjusting and checking the business process model. The main advantage of this approach is the fact that it can be applied to already running business processes.

An alternative approach focuses on the early design phases and takes a business process model and the compliance rules as input and automatically generates a business process model that is *compliant by design* [53], cf. Fig. 1.2. This has several advantages: First, a subsequent proof and potential corrections are not required. This may speed up the modeling process. Second, the approach is flexible as the generation can be repeated when rules are added, removed, or changed. Third, the approach is complete in the sense that an unsuccessful model generation can be interpreted as "the business process cannot be *made* compliant" rather than "the current model is not compliant". Fourth, compliance is not only detected, but actually enforced. That is, noncompliant behavior becomes technically impossible.

1.5.2.1 Modeling compliance rules

This subsection investigates to what extend compliance rules can be integrated into the artifact-centric approach. Before we present different shapes of compliance rules and their formalization with Petri nets, we first discuss the difference between a policy and a compliance rule.

Enforcing policies vs. monitoring compliance rules

As described in Sect. 1.4.2, we use policies to express interdependencies between artifacts and explicitly restrict behavior by making the firing of transitions impossible. Policies thereby express domain knowledge about the business process and its artifacts and are suitable to *inhibit* implausible or undesired behavior. This finally affects the subsequent controller synthesis.

In contrast, a compliance rule specifies behavior that is not under the direct control of the business process designer. Consequently, a compliance rule *must not restrict the behavior of the process, but only monitor it to detect noncompliance*. For instance, a compliance rule must not disable external choices within the business process as they cannot be controlled by any agent. If such a choice would be disabled to achieve compliance, the resulting business process model would be spurious as the respective choice could not be disabled in reality. Therefore, compliance rules must not restrict the behavior of the artifacts, but only restrict the final states of the model. This may classify behavior as undesired (viz. noncompliant), but this behavior remains reachable. Only if this behavior can be circumvented by the controller synthesis, we faithfully found a compliant business process which can be actually implemented. We formalize this nonrestricting nature as *monitor property* [40, 57]. Intuitively, this property requires that in every reachable marking of an artifact, it holds that for each action label of that artifact a transition with that label is activated. This rules out situations in which the firing of a transition in a composition is inhibited by a compliance rule.

Expressiveness of compliance rules

Conceptually, we model compliance rules by artifacts with the monitor property. Again, adding a compliance rule to an artifact-centric model boils down to composition, cf. Def 4. The monitor property ensures that the compliance rule's transitions are synchronized with the other artifacts, but without restricting (i. e., disabling)

actions. That is, the life cycle of a compliance rule model evolves together with the artifacts' life cycles, but may only affect the final states of the composed model.

In a finite-state composition of artifacts, the set of runs reaching a final state forms a regular language. The terminating runs of a compliance rule (i. e., sequences of transitions that reach a final marking) describe compliant runs. This set again forms a regular language. In the composition of the artifacts and the compliance rules, these regular languages are synchronized — viz. intersected — yielding a subset of terminating runs. Regular languages allow to express a variety of relevant scenarios. In fact, we can express all patterns listed by Dwyer et al. [19], including:

- enforcement and existence of actions (e. g., *"Every compliant run must contain an action 'archive claim'."*),
- absence/exclusion of actions (e. g., *"The action 'withdraw claim' must not be executed."*),
- ordering (precedence and response) of actions (e. g., *"The action 'create settlement' must be executed after 'submit claim', but before 'archive claim'."*), and
- numbering constraints/bounded existence of actions (e. g., *"The action "partially pay settlement" must not be executed more than three times".*).

The explicit model of data states of the artifacts further allows to express rules concerning data flow, such as:

- enforcement/exclusion of data states (e. g., *"The claim's state 'fraud reported' and the settlement's state 'paid' must never coincide."*), or
- data and control flow concurrence (e. g., "*The action 'publish review' may only be executed if the review artifact is in state 'reviewers blinded'*.").

Additionally, the explicit modeling of the location of the artifacts allows to express spacial constraints:

- enforcement/exclusion of actions at specific locations (e. g., *"The task 'sign' may only be executed if the contract artifact is at the human resources department*"), or
- enforcement/exclusion of the transport/transfer of an artifact in a certain state (e. g., *"iPhone prototypes must not leave the company premises after the operating system is installed."*).

On top of that, any combinations are possible, allowing to express complex compliance rules.

Limitations

Apart from the conceptual richness, the presented approach has some theoretical limits. First, it is not applicable to nonregular languages. For instance, a rule requiring that a compliant run must have an arbitrary large, but equal number of a and b actions or that a and b actions must be properly balanced (Dyck languages) cannot be expressed with a finite-state models. Second, rules that affect infinite runs (e. g., certain LTL formulae [50]) cannot be expressed. Infinite runs are predominantly used

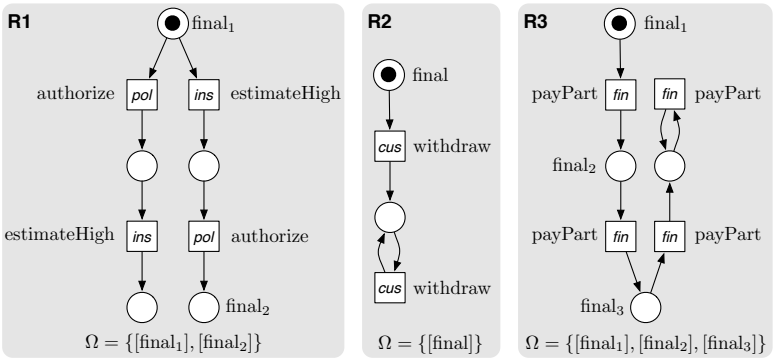


Fig. 1.9 Compliance rules modeled as Petri nets

to reason about reactive systems. A business process, however, is usually designed to eventually reach a final state — this basically is the essence of the soundness property. Therefore, we shall focus on an interpretation of LTL which only considers finite runs, similar to a semantics described by Havelund and Roşu [26]. Third, just like Awad et al. [8], we also do not consider the **X** (next state) operator of CTL*, because we typically discuss distributed systems in which states are not partially ordered. Forth, we do not use timed Petri nets and hence can make no statements on temporal properties of business processes. However, we can abstract the variation of time by events such as "time passes" or data states such as "expired" as in [24, 46].

Conceptually, an extension toward more expressive constraints would be possible. For instance, pushdown automata could be used to express Dyck languages, a yielding context-free language as product. However, this extension would need further theoretical consideration as, for instance, controllability is undecidable in case of infinite state systems [47].

Example formalizations of compliance rules

As mentioned earlier, we again use artifacts (i. e., Petri nets with final markings and action labels) that satisfy the monitor property to model compliance rules. As an example, we consider the following compliance rules for our example insurance claim process:

R1	*All insurance claims with an estimated high settlement must be authorized.*
R2	*Customers must not be allowed to withdraw insurance claims.*
R3	*Settlements should be paid in at most three parts.*

Figure 1.9 shows the Petri net formalizations of these compliance rules. In rule R1, we exploited the fact that the actions "authorize" and "estimateHigh" are executed at most once. In rule R2 and R3, the monitor property is achieved by allowing "withdraw" and "payPart" to fire in any reachable state. Without restriction of the

behavior, the final markings classify executions as compliant or not. For instance, executing "estimateHigh" in rule R1 without eventually executing "authorize" does not reach the final marking $[\text{final}_2]$. Other examples can be formalized similarly.

Discussion

We conclude this section by a discussion of the implications of using Petri nets to formalize compliance rules.

- *Single formalism.* We can model artifacts, policies, and compliance rules with the same formalism. Though we do not claim that Petri nets should be used by domain experts to model compliance regulations, using a single formalism still facilitates the modeling and verification process. Furthermore, each rule implicitly models compliant behavior which can be simulated. This is not possible if, for instance, arbitrary LTL formulae are considered.
- *Level of abstraction.* Rules can be expressed using minimal overhead. Each rule contains only those places and transitions that are affected by the rule and plus some additional places to model further causalities. In particular, no placeholder elements (e. g., anonymous activities in BPMN-Q [7]) are required. These placeholder elements must not be confused with "wildcard" dependencies, for instance requiring input from artifact A, B, or C. To formalize such dependencies with our artifact model, they need to be unfolded explicitly. Of course, syntactic extensions may be introduced to modeling languages to compact the models.
- *Independent design.* The rules can be formulated independently of the artifact and policy models. That is, the modeler does not need to be confronted with the composite model. This modular approach is more likely to scale, because the rules can also be validated independently of the other rules.
- *Reusability.* The composition is defined in terms of action labels. Therefore, rules may be reused in different business process models as long as the labels match. This can be enforced using standard naming schemes or ontologies.
- *Runtime monitoring.* The monitor property ensures that the detection of noncompliant behavior is transparent to the process as no behavior is restricted. Therefore, the models of the compliance rules can be also used to check compliance during or after runtime, for instance by inspecting execution logs.
- *Rule generation.* Finally, the structure of the Petri nets modeling compliance rules is very generic. Therefore, it should be possible to automatically generate Petri nets for standard scenarios or to provide templates to which only the names of the constrained actions need to be filled. Also, the monitor property can be automatically enforced.

1.5.2.2 Compliance by design

This section presents the second ingredient of this contribution: the construction of business process models that are compliant by design. Beside the construction, we also discuss the diagnosis of noncompliant business process models.

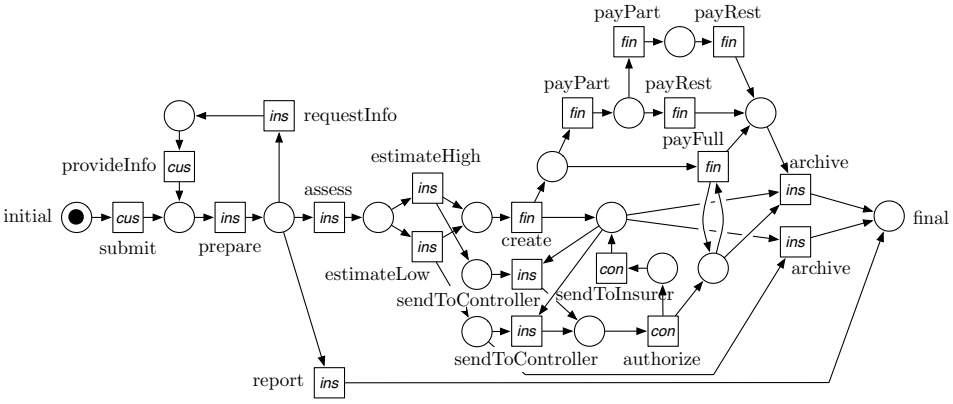


Fig. 1.10 Operational business process satisfying the compliance rules R1–R3

Constructing compliant models

None of the compliance rules discussed in the previous section hold in the example process depicted in Fig. 1.8. This noncompliance can be detected by standard model checking tools. They usually provide a counterexample which describes how a noncompliant situation can be reached. For instance, the action sequence "1. submit, 2. prepare, 3. requestInfo, 4. withdraw" is a witness that the process does not comply with rule R2 from Sect. 1.5.2.1. To satisfy this requirement, the transition "withdraw" can be simply removed. However, implementing the other rules is more complicated, and each modification would require another compliance check.

We propose to *synthesize* a compliant model instead of verifying compliance. By composing the Petri net models of the artifacts, the policies, and the compliance rules and by taking the goal states into account, we derive a Petri net that models the artifacts' life cycles that are restricted by the policies and whose final states are constrained by the goal states and the compliance rules. *Compliant behavior is now reduced to weak termination*, and we can apply the same algorithm [57] and tool [42] to synthesize a controller. If such a controller exists, it provides an operational model that specifies the order in which the agents need to perform their actions. This model is *compliant by design* — a subsequent verification is not required. Beside weak termination (and hence, compliance), the synthesis algorithm further guarantees the resulting model is *most permissive* [57]. That is, exactly that behavior has been removed that would violate weak termination. Another important aspect of the approach is its flexibility to add further compliance rules. That is, we do not need to edit the existing model, but we can simply repeat the synthesis for the new rule set.

Running example (cont.) Figure 1.10 depicts the resulting business process model. It obviously contains no transition labeled with "withdraw", but the implementation of the other rules yielded a whole different structure of the part modeling the settlement processing. *It is important to stress that the depicted business process model has been synthesized completely automatically* using the partner synthesis tool Wendy [42] and the Petri net synthesis tool Petrify [17]. Admittedly, it is a rather complicated model,

but any valid implementation of the compliance rules would yield the same behavior or a subset. Though our running example is clearly a toy example, experimental results [42] show that controller synthesis can be effectively applied to models with millions of states.

1.6 Related work

Artifact-centric approaches have recently received broad attention from both academia and industry. This section sketches related work in the context of several categories touched by this chapter.

GSM. The guard-stage-milestone model (GSM) [28, 29] is the result of IBM's long-standing effort of promoting the artifact-centric view on business processes. GSM follow a clearly declarative way which allows for greater flexibility. Milestones can be compared to our goal states. However, GSM has a stronger focus on modeling and further allows for elegant hierarchical models. Artifact-centric models are often formalized using infinite state systems (cf. [28, 18, 25]) whereas our approach heavily relies synthesis and hence on finite state models. GSM further assumes active artifacts that may perform service calls and has no concept of locations.

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. [12] 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. [14] and Fritz et al. [22] for first order logics. Gerede and Su [23] 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. [30] 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 of the aggregate. This centralized approach can be mimicked by our location-aware approach by remotely accessible immobile artifacts. However, to tackle potential problems arising with the hosting of sensible data in a centralized data hub, we propose to derive explicit message flow from an location-aware artifact model.

Execution. For the execution of artifact-centric processes exist different approaches: Li and Wu [34] assume that each artifact has a service interface and

propose a translation into WS-BPEL. They assume, however, an existing workflow that coordinates the artifact execution — such a workflow could be the result of our partner synthesis approach. Opposed to the transformation of the artifacts, Ngamakeur et al. [48] promote the direct execution of artifacts as a translation is less flexible and is potentially connected with information loss. In our approach, we also do not change the artifacts, but only derive a workflow that coordinates the executing agents. Finally, Liu et al. [35] translate the ECA tules of Artiflow to WS-BPEL.

Conformance. Fahland et al. [21] extend conformance checking to artifact-centric business processes. Their approach bases on comparing recorded data base states with a process model and can be applied similarly to our model. Yongchareon et al. [58] study private and shared artifacts. Based on this distinction, public and private views can be defined. To support refinement of executable process models, we rely on existing results [2, 3, 56] which need to be extended toward an artifact-centric model.

Proclets. Aalst et al. [1, 4] 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.

Compliance by detection. Awad et al. [8] investigate a pattern-based compliance check based on BPMN-Q [7]. They also cover the compliance rule classes defined by Dwyer et al. [19] and give a CTL formalization as well as an antipattern for each rule. These antipatterns are used to highlight the compliance violations in a BPMN model. Such a visualization is very valuable for the process designer and it would be interesting to see whether such antipatterns are also applicable to the artifact-centric approach. Sadiq et al. [54] use a declarative specification of compliance rules from which they derive compliance checks. These checks are then annotated to a business process and monitored during its execution. These checks are similar to the nonblocking compliance rule models that only monitor behavior rather than constraining it. Lu et al. [46] compare business processes with compliance rules and derive a compliance degree. This is an interesting approach, because it replaces yes/no answers by numeric values which could help to easier diagnose noncompliance. Knuplesch et al. [31] analyze data aspects of operational business process models. Similar to the artifact-centric approach, data values are abstracted into compact life cycles.

Compliance by design. Goedertier and Vanthienen [24] introduce the declarative language PENELOPE to specify compliance rules. From these rules, a state space and a BPMN model is generated which is compliant by design. This approach is limited to acyclic process models. Furthermore, the purpose of the generated model is rather the validation of the specified rules than the execution. Küster et al. [33] study the interplay between control flow models and object life cycles. The authors present an algorithm to automatically derive a sound process model from given object life cycles. The framework is, however, not designed to express dependencies between life cycles and therefore cannot specify complex policies or compliance rules.

1.7 Conclusion

We extended the idea of artifact-centric process design to an interorganizational setting where artifacts cannot be 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 in a corresponding activity-centric process. 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. We can further appropriately integrate compliance requirements. 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 an activity-centric models 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 [20, 9]. 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 global activity-centric model 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., [11, 13, 56]). In [3], 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. The approach requires a suitable decision procedure for checking accordance [56, 55] or powerful transformation rules which preserve accordance [32]. 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 [44, 45]. 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. Adding results from [57] 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 [57]. 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. Furthermore, a reverse translation from activity-centric to artifact-centric models could be a promising direction of research. This could promote the artifact-centric modeling style to an equitable view on the process under investigation.

The main practical limitation is the lack of a proper modeling language, because the presented Petri net formalization is only a conceptual modeling language. We recently developed an extension [41] for BPMN [49] to provide a graphical notation that is more accessible for domain experts to model artifacts, policies, and compliance rules. A canonic next step would then be the integration of the approach into a modeling tool and an empirical evaluation thereof. Furthermore, we concentrated on the early design of a business process and did not consider its execution. For business processes that are already in execution, our approach is currently not applicable as the translations from conceptual models to industrial languages are still very immature [39].

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., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: From public views to private views — correctness-by-design for services. In: WS-FM 2007, LNCS 4937, pp. 139–153. Springer (2008)
3. 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 (2010)
4. 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)
5. Aalst, W.M.P.v.d., Pesic, M.: DecSerFlow: Towards a truly declarative service flow language. In: WS-FM 2006, LNCS 4184, pp. 1–23. Springer (2006)
6. Alistair Barros, M.D., ter Hofstede, A.: Service interaction patterns. In: BPM 2005, vol. LNCS 3649, pp. 302–318. Springer Verlag (2005)
7. Awad, A.: BPMN-Q: a language to query business processes. In: EMISA 2007, LNI P-119, pp. 115–128. GI (2007)
8. Awad, A., Weidlich, M., Weske, M.: Visually specifying compliance rules and explaining their violations for business processes. J. Vis. Lang. Comput. **22**(1), 30–55 (2011)
9. Ball, T., Chaki, S., Rajamani, S.K.: Parameterized verification of multithreaded software libraries. In: TACAS 2001, LNCS 2031, pp. 158–173. Springer (2001)
10. Ben-Ari, M., Manna, Z., Pnueli, A.: The temporal logic of branching time. In: POPL '81, pp. 164–176. ACM (1981)
11. Benatallah, B., Casati, F., Toumani, F.: Representing, analysing and managing Web service protocols. Data Knowl. Eng. **58**(3), 327–357 (2006)

12. Bhattacharya, K., Gerede, C.E., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: BPM 2007, LNCS 4714, pp. 288–304. Springer (2007)
13. Bravetti, M., Zavattaro, G.: Contract based multi-party service composition. In: FSEN 2007, LNCS 4767, pp. 207–222. Springer (2007)
14. Calvanese, D., Giacomo, G.D., Hull, R., Su, J.: Artifact-centric workflow dominance. In: ICSOC/ServiceWave 2009, LNCS 5900, pp. 130–143. Springer (2009)
15. 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)
16. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
17. Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: A tool for manipulating concurrent specifications and synthesis of asynchronous controllers. Trans. Inf. and Syst. **E80-D**(3), 315–325 (1997)
18. Damaggio, E., Hull, R., Vaculín, R.: On the equivalence of incremental and fixpoint semantics for business artifacts with guard-stage-milestone lifecycles. In: BPM 2011, LNCS 6896, pp. 396–412. Springer (2011)
19. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: ICSE 1999, pp. 411–420. IEEE (1999)
20. Emerson, E.A., Kahlon, V.: Reducing model checking of the many to the few. In: CADE 2000, LNCS 1831, pp. 236–254. Springer (2000)
21. Fahland, D., de Leoni, M., van Dongen, B.F., van der Aalst, W.M.P.: Behavioral conformance of artifact-centric process models. In: BIS 2011, LNBIP 87, pp. 37–49. Springer (2011)
22. Fritz, C., Hull, R., Su, J.: Automatic construction of simple artifact-based business processes. In: ICDT 2009, ACM International Conference Proceeding Series 361, pp. 225–238. ACM (2009)
23. Gerede, C.E., Su, J.: Specification and verfication of artifact behaviors in business process models. In: ICSOC 2007, LNCS 4749, pp. 181–192. Springer (2007)
24. Goedertier, S., Vanthienen, J.: Designing compliant business processes with obligations and permissions. In: BPM Workshops 2006, LNCS 4103, pp. 5–14. Springer (2006)
25. Hariri, B.B., Calvanese, D., Giacomo, G.D., Masellis, R.D., Felli, P.: Foundations of relational artifacts verification. In: BPM 2011, LNCS 6896, pp. 379–395. Springer (2011)
26. Havelund, K., Roşu, G.: Testing linear temporal logic formulae on finite execution traces. Technical Report 01.08, RIACS (2001)
27. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley (2003)
28. Hull, R., Damaggio, E., Fournier, F., Gupta, M., Heath, F.T., Hobson, S., Linehan, M.H., Maradugu, S., Nigam, A., Sukaviriya, P., Vaculín, R.: Introducing the guard-stage-milestone approach for specifying business entity lifecycles. In: WS-FM 2010, LNCS 6551, pp. 1–24. Springer-Verlag (2011)
29. Hull, R., Damaggio, E., Masellis, R.D., Fournier, F., Gupta, M., Heath, F.T., Hobson, S., Linehan, M.H., Maradugu, S., Nigam, A., Sukaviriya, P.N., Vaculín, R.: Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In: DEBS 2011, pp. 51–62 (2011)
30. Hull, R., Narendra, N.C., Nigam, A.: Facilitating workflow interoperation using artifact-centric hubs. In: ICSOC/ServiceWave 2009, pp. 1–18 (2009)
31. Knuplesch, D., Ly, L.T., Rinderle-Ma, S., Pfeifer, H., Dadam, P.: On enabling data-aware compliance checking of business process models. In: ER 2010, LNCS 6412, pp. 332–346. Springer (2010)
32. 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 (2008)
33. Küster, J.M., Ryndina, K., Gall, H.: Generation of business process models for object life cycle compliance. In: BPM 2007, LNCS 4714, pp. 165–181. Springer (2007)
34. Li, D., Wu, Q.: Translating artifact-based business process model to BPEL. In: CSEE (2), *Communications in Computer and Information Science*, vol. 215, pp. 482–489. Springer (2011)
35. Liu, G., Liu, X., Qin, H., Su, J., Yan, Z., Zhang, L.: Automated realization of business workflow specification. In: ICSOC/ServiceWave Workshops, LNCS 6275, pp. 96–108. Springer (2009)

36. Lohmann, N.: Why does my service have no partners? In: WS-FM 2008, LNCS 5387, pp. 191–206. Springer (2009)
37. Lohmann, N.: Compliance by design for artifact-centric business processes. In: BPM 2011, LNCS 6896, pp. 99–115. Springer (2011)
38. Lohmann, N.: Compliance by design for artifact-centric business processes. Inf. Syst. (2012). (Accepted for publication in March 2012)
39. Lohmann, N., Kleine, J.: Fully-automatic translation of open workflow net models into simple abstract BPEL processes. In: Modellierung 2008, *Lecture Notes in Informatics (LNI)*, vol. P-127, pp. 57–72. GI (2008)
40. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral constraints for services. In: BPM 2007, LNCS 4714, pp. 271–287. Springer (2007)
41. Lohmann, N., Nyolt, M.: Artifact-centric modeling using BPMN. In: ICSOC 2011 Workshops, *LNCS*, vol. 7221, pp. 54–65. Springer (2012)
42. Lohmann, N., Weinberg, D.: Wendy: A tool to synthesize partners for services. In: PETRI NETS 2010, LNCS 6128, pp. 297–307. Springer (2010). Tool available at `http://service-technology.org/wendy`.
43. Lohmann, N., Wolf, K.: Artifact-centric choreographies. In: ICSOC 2010, LNCS 6470, pp. 32–46. Springer-Verlag (2010)
44. Lohmann, N., Wolf, K.: Realizability is controllability. In: WS-FM 2009, LNCS 6194, pp. 110–127. Springer (2010)
45. Lohmann, N., Wolf, K.: Decidability results for choreography realization. In: ICSOC 2011, LNCS 7084, pp. 92–107. Springer-Verlag (2011)
46. Lu, R., Sadiq, S.W., Governatori, G.: Compliance aware business process design. In: BPM 2007 Workshops, LNCS 4928, pp. 120–131. Springer (2007)
47. Massuthe, P., Serebrenik, A., Sidorova, N., Wolf, K.: Can I find a partner? Undecidablity of partner existence for open nets. Inf. Process. Lett. **108**(6), 374–378 (2008)
48. Ngamakeur, K., Yongchareon, S., Liu, C.: A framework for realizing artifact-centric business processes in service-oriented architecture. In: DASFAA 2012, LNCS 7238, pp. 63–78. Springer (2012)
49. OMG: Business Process Model and Notation (BPMN). Version 2.0, Object Management Group (2011). URL `http://www.omg.org/spec/BPMN/2.0`
50. Pnueli, A.: The temporal logic of programs. In: FOCS 1977, pp. 46–57. IEEE (1977)
51. Pnueli, A.: In transition from global to modular temporal reasoning about programs. In: Logics and models of concurrent systems, volume F-13 of NATO Advanced Summer Institutes, pp. 123–144. Springer (1985)
52. Ryndina, K., Küster, J.M., Gall, H.: Consistency of business process models and object life cycles. In: MoDELS Workshops, LNCS 4364, pp. 80–90. Springer (2006)
53. Sackmann, S., Kähmer, M., Gilliot, M., Lowis, L.: A classification model for automating compliance. In: CEC/EEE 2008, pp. 79–86. IEEE (2008)
54. Sadiq, S.W., Governatori, G., Namiri, K.: Modeling control objectives for business process compliance. In: BPM 2007, LNCS 4714, pp. 149–164. Springer (2007)
55. 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)
56. 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)
57. Wolf, K.: Does my service have partners? LNCS T. Petri Nets and Other Models of Concurrency **5460**(2), 152–171 (2009)
58. Yongchareon, S., Liu, C., Zhao, X.: An artifact-centric view-based approach to modeling inter-organizational business processes. In: WISE 2011, LNCS 6997, pp. 273–281. Springer (2011)