Execution Semantics for Process Choreographies with Data

Tom Lichtenstein(✉) and Mathias Weske

Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
{Tom.Lichtenstein,Mathias.Weske}@hpi.de

Abstract. Interaction models, such as BPMN choreography diagrams, enable the coordination of interactions between organizations within a process choreography. Since choreography participants typically do not share a central data store, data flow must be considered during design to ensure that each participant has sufficient data to continue a conversation. However, current choreography modeling languages lack a concise notation and execution semantics for data exchange. This paper refines the data flow specifications of choreography diagrams using supplemental models. Furthermore, the execution semantics for data exchange is formally defined by data-enhanced interaction Petri nets. The extended semantics enable the analysis of data awareness and data consistency at design time.

Keywords: Process choreographies · Interaction Petri nets · Data flow

1 Introduction

In a networked economy, the exchange of goods and services between organizations is key to success. To ensure effective collaboration, the interactions of the organizations' internal processes must be carefully designed, e.g., in process choreographies. Since choreography participants typically do not share a central data store, interorganizational data flow must be considered to ensure that each participant has sufficient data to advance the conversation. While interaction models such as BPMN choreography diagrams [15] allow us to define interaction behavior from a global perspective, current choreography modeling languages lack concise notation and execution semantics for data exchange. As a result, it is not possible to verify at design time that message senders are aware of the required data and that participants affected by global decisions have a consistent view of the data driving those decisions. This paper aims to improve data modeling support for process choreographies by refining the specification of exchanged data for choreography diagrams. A formal execution semantics for data exchange is provided by mapping supplemented choreography diagrams to interaction Petri nets extended with a data perspective. The mapping provides a foundation for a data-aware formal analysis and verification of choreographies at design time.

C. Di Francescomarino et al. (Eds.): BPM 2023 Forum, LNBIP 490, pp. 90–106, 2023.
https://doi.org/10.1007/978-3-031-41623-1_6

The remainder of the paper is organized as follows: Sect. 2 outlines the fundamentals of process choreographies and interaction Petri nets, followed by a motivating example in Sect. 3. Section 4 introduces the refined data flow specifications and formal execution semantics for data exchange as the main contributions of this paper. Next, Sect. 5 discusses the presented approach, and Sect. 6 gives a brief overview of related work. Finally, Sect. 7 summarizes the results of this work.

2 Preliminaries

As a foundation for this work, this section outlines the basic concepts and formal principles of process choreographies in Sect. 2.1 and interaction Petri nets in Sect. 2.2.

2.1 Process Choreographies

Process choreographies define the possible interaction sequences between business actors (i.e., *choreography participants*) that collaborate to achieve a goal [5]. Each choreography participant is associated with a *role*. An execution of a choreography is referred to as a *conversation*. Definition 1 specifies the main concepts of process choreographies.

Definition 1. *(Process Choreography). A process choreography is defined by a tuple* $\mathcal{C} = (N, SF, R, M, \mathcal{G}, grd, init, resp, msg)$*, where:*

- $N \subseteq \mathcal{T} \times G \times E$ *is a finite, non-empty set of nodes including choreography tasks* $\mathcal{T}$*, gateways* G*, and events* E*,*
- E *can be partitioned into disjoint sets of start events* E_s *and end events* E_e*,*
- G *can be partitioned into disjoint sets of event-based gateway splits* G^s_e*, exclusive gateway splits* $G^s_\times$*, exclusive gateway joins* $G^j_\times$*, parallel gateway splits* G^s_+*, and parallel gateway joins* G^j_+*,*
- $SF \subseteq N \times N$ *is a finite, non-empty set of sequence flows,*
- R *is a finite, non-empty set of participant roles,*
- M *is a finite set of messages,*
- $\mathcal{G}$ *is a finite set of guards,*
- $grd : G^s_\times \times N \to \mathcal{G}$ *assigns a guard to a sequence flow,*
- $init : \mathcal{T} \to R$ *assigns the initiating role to a choreography task,*
- $resp : \mathcal{T} \to R$ *assigns the respondent role to a choreography task, and*
- $msg : \mathcal{T} \to (M \times M) \cup M \cup \{\emptyset\}$ *assigns messages to a choreography task.*

BPMN 2.0 introduces *choreography diagrams* as an interaction modeling language for process choreographies [15]. Unlike BPMN collaboration diagrams, choreography diagrams abstract from process-internal details and focus only on interorganizational behavior. Choreography diagrams represent interactions as *choreography tasks*, hereafter referred to as tasks. Each task is associated with an initiator (white badge) and a respondent (gray badge). Optionally, an initial message (white envelope) and a response message (gray envelope) can be

specified. A task with a response message is considered a two-way interaction involving a request from the initiator and a response from the respondent.

Similar to BPMN collaboration diagrams, sequence flow arcs specify order dependencies between tasks. In addition, gateways allow the specification of exclusive and parallel behavior. Sequence flow arcs originating from an exclusive gateway can be associated with a guard that specifies the condition for continuing along that path. Note that all participants affected by the decision must have the same view of the data on which the decision is based [15]. In contrast to process orchestrations, choreographies do not assume a central data store, as each participant typically maintains its data locally. Data can only be exchanged via messages [12]. An example of a choreography diagram is depicted in Fig. 1.

2.2 Interaction Petri Nets

Models facilitate development and the exchange of ideas among experts, yet precise semantics are essential for their implementation and analysis. In business process modeling, *Petri nets* are widely used to provide concise execution semantics [7]. Petri nets consist of places and transitions connected by directed arcs. Places may contain tokens. If all places connected with an incoming arc contain tokens, a transition can be fired to consume tokens from the incoming places and produce tokens in the outgoing places. A distribution of tokens to places is referred to as a *marking* [1]. Decker et al. introduce *Interaction Petri Nets* (IPN) as an extension of Petri nets for describing interaction models [5]. IPNs represent each interaction by a single transition labeled with the initiator, the respondent, and a description of the message. The additional information allows reasoning about enforceability aspects of an interaction model [4]. A firing sequence of transitions represents a conversation. Based on definitions from the literature [5,10], we define interaction Petri nets as follows:

Definition 2. *(Interaction Petri net). An interaction Petri net is defined by a tuple* $\mathcal{I} = (P, T, F, R, init, resp, m_0)$*, where*

- P *is a finite set of places,*
- T *is a finite set of transitions, which can be partitioned into disjoint sets of interactions* T_I*, events* T_E*, and silent transitions* T_S*,*
- $F \subseteq (P \times T) \cup (T \times P)$ *is a finite set of arcs,*
- R *is a finite set of roles,*
- $init : T_I \rightarrow R$ *assigns an initiating role to an interaction transition,*
- $resp : T_I \rightarrow R$ *assigns a respondent to an interaction transition, and*
- $m_0 : P \rightarrow \mathbb{N}$ *assigns the initial number of tokens to each place, thus specifying an initial marking for the net.*

3 Motivating Example

To illustrate the need for concise data semantics for choreographies, in this section we present an example choreography inspired by the shipment of goods

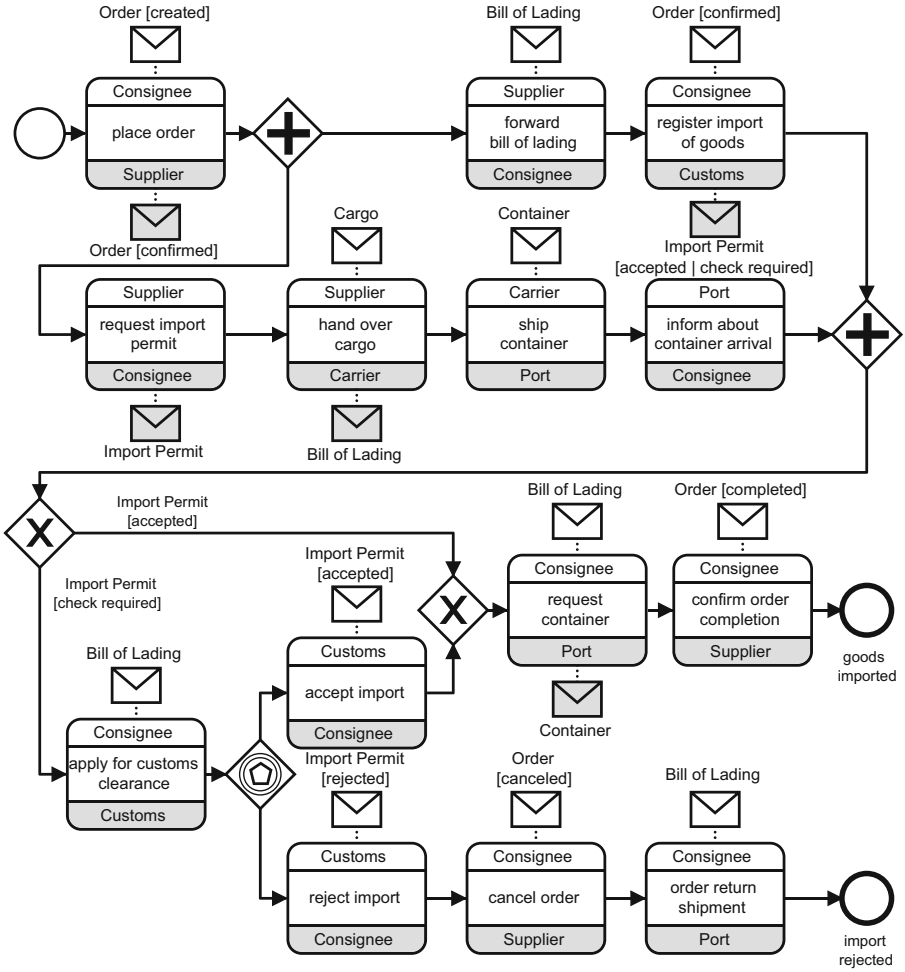


Fig. 1. Choreography diagram describing the international transport of goods by ship between a consignee and a supplier, considering customs.

by sea to a European Union member state. The choreography diagram shown in Fig. 1 illustrates the key steps, including the consignee ordering goods from a supplier, the supplier sending the container of goods to the destination port via a carrier, and customs inspecting the import of the goods. For each order, the supplier requests the import permit from the consignee, which must be issued by customs before the container is handed over to the carrier. Upon arrival, the consignee can access the container using the bill of lading issued by the carrier. If the import permit requires further checks, the consignee must first apply for customs clearance. If customs rejects the import, the container must be returned.

When data is not considered, the choreography diagram meets the local enforceability requirements [4]. However, considering data exchange raises enforceability concerns. In particular, after the order is confirmed by the supplier, the subsequent steps require the supplier to forward the bill of lading and the consignee to provide the import permit. Both documents, however, are issued by the carrier or customs respectively and are sent only in subsequent tasks, resulting in a deadlock. In addition, the BPMN 2.0 standard requires that participants affected by an exclusive gateway must share the same view of the data on which the decision is based on [15]. After the container arrived, the port has not received the import permit and therefore is unaware of whether a return shipment must be expected. Therefore, the design of the choreography raises concerns about data exchange regarding:

- Data awareness: Is the sender of a message aware of the required data?
- Data consistency: Do participants have the same view of the exchanged data?
- Data dependencies: What dependencies exist between the exchanged data?

Since erroneous data flow may not be obvious in complex choreographies, precise specifications and semantics for interorganizational data exchange are required to enable the analysis of process choreographies with data at design time.

4 Execution Semantics for Choreographies with Data

Specifying and analyzing interorganizational data flow requires extending the execution semantics and refining the notion of choreography diagrams. In the following, Sect. 4.1 introduces supplementary models for specifying data exchange in choreography diagrams. In addition, Sect. 4.2 presents data-enhanced interaction Petri nets as a formal basis for defining execution semantics for data exchange. Finally, Sect. 4.3 proposes a mapping of supplemented choreography diagrams to data-enhanced interaction Petri nets to define execution semantics for choreography diagrams with data specifications.

4.1 Data Exchange Specifications for Choreography Diagrams

Choreography diagrams allow only limited specification of data exchange. Message elements can be assigned labels to describe the content of the message, but no clear semantics are provided for the labels, which can lead to different interpretations of the behavior. In this section, we refine choreography diagrams by introducing more concise data exchange specifications for message elements. To remain compliant with the BPMN standard, the notation of the choreography diagrams is not adapted, which facilitates modeling with existing tools. Instead, the specification is composed of supplemental models that, in addition to the choreography diagram, define the behavior and relations between the exchanged data. The additional models consist of a *shared data model* and *distributed object lifecycle models*. Both types of models and their relationship to choreography diagrams are described below.

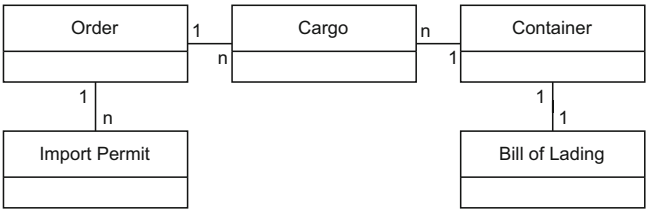


Fig. 2. A shared data model defining the message types and their relations for messages exchanged in the exemplary shipping choreography.

Shared Data Model. A shared data model is used to globally specify the types of messages exchanged in a choreography. Each class represents one message type. An instance of a class, referred to as a *message object*, is considered a unit of data, such as a document, that can be exchanged via a message associated with the corresponding class. While messages can also refer to physical items, such as a container or cargo, message objects refer only to a virtual representation of these items. Consequently, when a message is sent, the message object is not lost to the sender, but both the sender and receiver are aware of the message object.

Definition 3. *(Shared Data Model). A shared data model is defined by a tuple* $\mathcal{D} = (C, \mathcal{R}, mult)$*, where:*

- C *is a non-empty, finite set of classes,*
- $\mathcal{R} \subseteq \{(c_1, c_2) \mid c_1, c_2 \in C \wedge c_1 \neq c_2\}$ *is a symmetric relation of classes,*
- $mult : \mathcal{R} \to \{(1,1), (1,n), (n,1), (n,m)\}$ *assigns a multiplicity to a relation.*

Similar to the approach proposed by Meyer et al. [12], the data model is shared by all participants so that each participant has the same understanding of the types of messages that can be exchanged. Figure 2 illustrates a shared data model for the exemplary shipping choreography. To reference the message types in the choreography diagram, the message elements of tasks can be annotated with the label of the class, as illustrated in Fig. 1.

As stated in Definition 3, the shared data model also defines relations with multiplicities between classes. To limit complexity, we consider only *one-to-one* $(1,1)$, *many-to-one* $(n,1)$, and *many-to-many* (n,m) relations, where it is expected that m and n can be zero. Relations can constrain the creation of message objects by implying dependencies. For example, according to Fig. 2, an 'Import Permit' cannot exist without an existing 'Order' due to their many-to-one relation: $mult((ImportPermit, Order)) = (n,1)$. Therefore, an 'Import Permit' object can only be instantiated if an 'Order' object already exists, which limits the possible behavior of the choreography. Accordingly, 'Container' and 'Bill of Lading' message objects must be created simultaneously to ensure the one-to-one relation. Many-to-many relations do not affect the creation of message objects, since it is expected that the objects can exist individually, given the assumption that n and m can be zero.

Distributed Object Lifecycle. During a conversation, message objects can be created or their contents changed. Similar to data states of data objects in BPMN process diagrams [15], we use *message states*, hereinafter referred to as *states*, to reflect the content of a message object. A state provides an abstract view of the content of a message object that is relevant to the business case under consideration. For example, an 'Order' message object can be in the states 'created', 'confirmed', 'completed', or 'canceled'. The mapping between states and actual attribute values is beyond the scope of this paper. Given a class $c \in C$, S_c denotes the set of possible states that a message object of c can be in. We refer to a message object of class $c \in C$ in a particular state $s \in S_c$ by the notion $c[s]$. Each class is considered to have at least one state.

To describe the possible states and allowed state transitions for a class in a choreography, we introduce *distributed object lifecycles*. As stated in Definition 4, distributed object lifecycles extend object lifecycles by specifying which role can perform which state transitions, since in choreographies message objects may be manipulated by different participants. For example, a consignee can create an 'Order' message object in the state 'created' but only the supplier can change the state to 'confirmed' as illustrated in Fig. 3. Each state is represented by a label in a circle, and allowed state transitions are represented by directed arcs associated with roles that can perform the transition. In addition to defining state transitions, distributed object lifecycles also constrain message object creation, since objects can only be created in initial states associated with an ingoing arc without an originating state. Furthermore, only roles associated with an initial state can create new message objects in the corresponding state. Similar to the shared data model, distributed object lifecycles are available to all participants.

Definition 4. *(Distributed Object Lifecycle). Let $\mathcal{S}$ be the universe of all possible states, a distributed object lifecycle of a class $c \in C$ is a finite state machine defined by a tuple $\mathcal{L}_c = (S_c, S_c^i, \delta_c, R, role)$, where:*

- *$S_c \subseteq \mathcal{S}$ is a non-empty, finite set of states associated with c,*
- *$S_c^i \subseteq S_c$ is a non-empty, finite set of initial states,*
- *$\delta_c \subseteq S_c \times S_c$ is a finite set of state transitions,*
- *R is a non-empty, finite set of roles, and*
- *$role : S_c^i \cup \delta_c \rightarrow R$ assigns a role to an initial state or a state transition.*

A message object can only be in one state for one participant at a time. It is assumed that if multiple participants are aware of a message object in the same state, these participants have the same view of its content. However, it should be noted that creating or modifying message objects is a local operation. Only when a message object is sent via a task, the respondent will receive the message object in the corresponding state. As a result, participants may have different views of a message object during a conversation.

States can also serve as constraints on interactions, since some interactions may require a message object to be sent in a particular state. To incorporate the constraints into choreography diagrams, states can be added to the class specifications of the messages using the notion $c[s]$ mentioned above. If a task

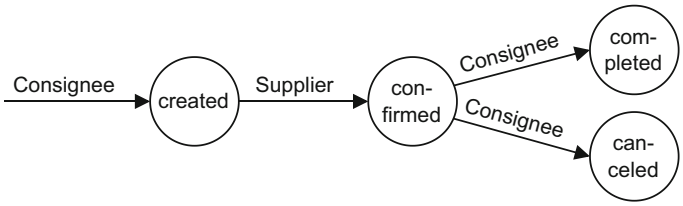


Fig. 3. A distributed object lifecycle that describes the allowed state transitions for message objects of the 'Order' class during a conversation. Annotations on arcs specify the roles that can perform the transitions.

accepts a message object in multiple states, all allowed states can be listed following the notion $c[s_1|...|s_n]$. Therefore, considering Fig. 1, the response of the task 'register import of goods' allows the sending of an 'Import Permit' message object in the state 'accepted' or 'check required'. The constraint implies further that the sender of the message must be aware of the message object in a corresponding state. If no state is specified, all possible states are accepted.

Furthermore, states can be used as guards for paths that follow exclusive gateways, as shown in Fig. 1. Thus, given a choreography $\mathcal{C}$, a guard is expected to specify a message object in an appropriate state required to continue with the associated path: $\mathcal{G} \subseteq C \times S$. For exclusive gateways, it is essential that all participants affected by the gateway have the same view of the data on which the decision is based. The identification of affected participants is discussed in more detail in Sect. 4.3.

The supplemental models introduced in this section can be used to specify exchanged data and data dependencies. However, to ensure that a choreography maintains data awareness and data consistency at design time, concise execution

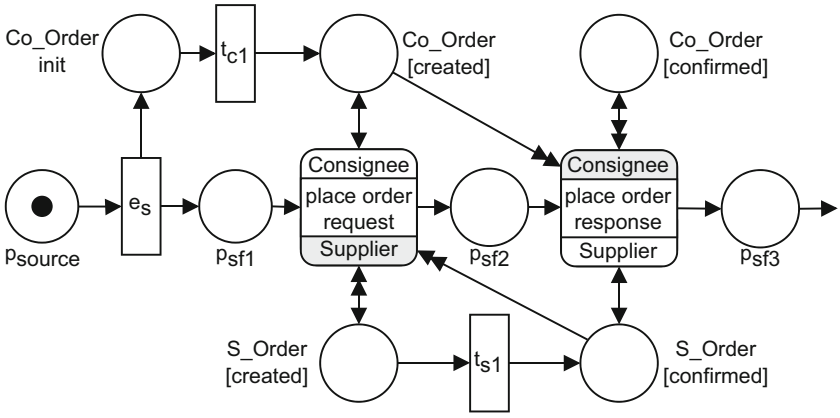


Fig. 4. Data-enhanced interaction Petri net depicting an excerpt of the shipping choreography including the start event and a two-way choreography task.

semantics are required. The next section introduces an extension to interaction Petri nets, providing a formal basis to define execution semantics for the data exchange specifications.

4.2 Data-Enhanced Interaction Petri Nets

The formal representation of data exchange in choreographies requires the extension of IPNs with additional elements. The extended net, referred to as *data-enhanced interaction Petri net*, is defined as follows:

Definition 5. *(Data-enhanced Interaction Petri Net). A data-enhanced interaction Petri net is defined by a tuple* $\mathcal{I}' = (P', T', F', RF, R, init, resp, m_0)$*, where*

- P' *is a finite set of places that can be partitioned into disjoint sets of control flow places* P*, message places* P_M*, and message initialization places* P_{MI}*,*
- T' *is a finite set of transitions that can be partitioned into disjoint sets of interactions* T_I*, events* T_E*, silent control flow transitions* T_S*, and silent message modification transitions* T_M*,*
- $F' \subseteq (P' \times T') \cup (T' \times P')$ *is a finite set of arcs that can be partitioned into sequence flow arcs* F *and message flow arcs* F_M*,*
- $RF \subseteq P_M \times T_I$ *is a finite set of reset arcs, and*
- $(P, T_I \cup T_E \cup T_S, F, R, init, resp, m_0)$ *is an interaction Petri net.*

The extension supports the representation of message objects using additional message places P_M. Each message place is dedicated to a participant and a message object in a particular state. A token in a message object place indicates that the corresponding participant is aware of the message object. To limit the creation of message objects, initialization places P_{MI} are introduced. Additional silent transitions T_M allow local creation and modification of message objects according to the message flow arcs F_M. Message flow arcs can also connect message places with interaction transitions to specify data exchange.

In addition, *reset arcs RF* are introduced, which set the number of tokens at all associated places to zero once the corresponding transition fires. Reset arcs do not constrain the firing of transitions. Thus, transitions can be fired even if they are connected to an empty place by a reset arc [1]. In the model, reset arcs are depicted with double arrowheads.

An example of a data-enhanced IPN is shown in Fig. 4. Similar to [10], the notion of interaction transitions is adapted to the contemporary notion of choreography tasks. The initiator of a task is indicated by a white badge and the respondent by a gray badge.

4.3 From Choreographies to Data-Enhanced Interaction Petri Nets

To define the execution semantics for data exchange in choreographies, we map choreography diagrams, supplemented with a shared data model and distributed

data object lifecycles, to data-enhanced interaction Petri nets. The mapping requires preprocessing the choreography diagram so that each task containing a response message is split into two tasks connected by a sequence flow arc. The result is semantically equivalent, with the first task representing the initial message and the second task representing the response. Consequently, after preprocessing, each task is associated with only one or zero message elements. We define the auxiliary function $obj : \mathcal{T} \rightarrow 2^{(C \times \mathcal{S})}$ to map each task to the corresponding class and states of the allowed message objects, as specified by the message element labels.

Hence, given a process choreography $\mathcal{C} = (N, SF, R, M, \mathcal{G}, grd, init, resp, msg)$, a shared data model $\mathcal{D} = (C, \mathcal{R}, mult)$, and a distributed object lifecycle $\mathcal{L}_c = (S_c, S_c^i, \delta_c, R, role)$ for each class $c \in C$, the models can be mapped to a data-enhanced interaction Petri net $\mathcal{I}' = (P', T', F', RF, R, init, resp, m_0)$ to represent the execution semantics as follows:

The set of roles R is taken from the choreography diagram. Correspondingly, the functions *init* and *resp* map the same roles for an interaction transition T_I as for the corresponding interaction $\mathcal{T}$ in the choreography. The mapping of the control flow semantics essentially follows the mappings provided in [5, 10]. However, unlike their mappings, multiple transitions are created for tasks that allow sending message objects in different states, since each transition is intended to transfer only one message object in one state, as depicted in Fig. 5. Since the additional transitions represent alternative executions of the same task with the same initiator and respondent, the extension does not affect local enforceability constraints. The set of silent transitions T_S is divided into disjoint

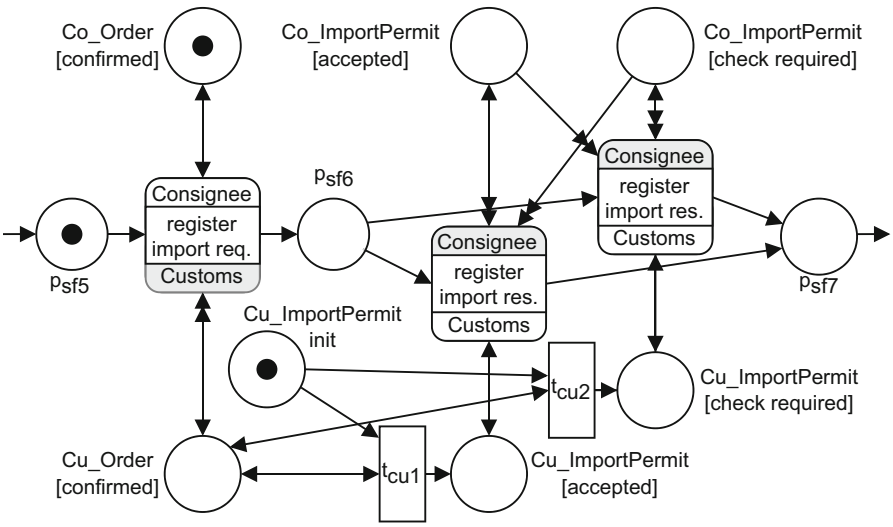


Fig. 5. Excerpt from a data-enhanced interaction Petri net representing the shipping example, illustrating the creation and sending of an 'Import Permit' message object in one of the allowed states using a predefined marking.

sets of exclusive gateways $T_\times$ and parallel gateways T_+. The mapping of the control flow to transitions and places is defined as follows:

$$
\begin{aligned}
P &= \{p_{source}, p_{sink}\} \cup \{p_{(n_1,n_2)} \mid (n_1,n_2) \in SF \wedge n_1 \notin G^s_e\} \\
T_E &= \{t_e \mid e \in E\} \\
T_I &= \{t_i \mid i \in \mathcal{T} \wedge obj(i) = \emptyset\} \cup \{t_{(i,c,s)} \mid i \in \mathcal{T} \wedge (c,s) \in obj(i)\} \\
T_\times &= \{t_{(g,n)} \mid g \in G^s_\times \wedge (g,n) \in SF\} \cup \{t_{(n,g)} \mid g \in G^j_\times \wedge (n,g) \in SF\} \\
T_+ &= \{t_g \mid g \in G^s_+ \cup G^j_+\} \\
m_0 &= \{[p_{source}]\}
\end{aligned}
$$

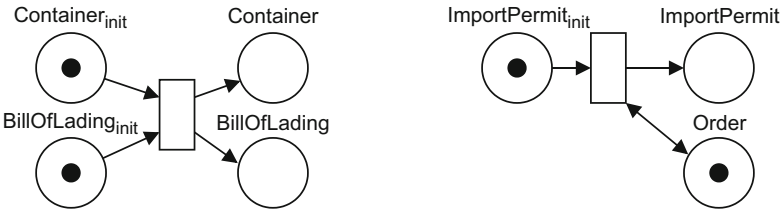
The mapping adds a source and a sink place that represent the start and end of a conversation. Only the source place contains a token in the initial marking. In addition, transitions and places are connected by arcs, enforcing the semantics of sequence flows and gateways:

$$
\begin{aligned}
F = {} & \{(p_{source}, t_e) \mid e \in E_s\} \cup \{(t_e, p_{sink}) \mid e \in E_e\} \cup \\
& \{(p_{(n_1,n_2)}, t_{n_2}) \mid (n_1,n_2) \in SF \wedge n_1 \notin G^s_e \wedge \\
& \quad (n_2 \in (E_e \cup G^s_+ \cup G^j_+) \vee (n_2 \in \mathcal{T} \wedge obj(n_2) = \emptyset))\} \cup \\
& \{(p_{(n,i)}, t_{(i,c,s)}) \mid (n,i) \in SF \wedge n \notin G^s_e \wedge i \in \mathcal{T} \wedge (c,s) \in obj(i)\} \cup \\
& \{(t_{n_1}, p_{(n_1,n_2)}) \mid (n_1,n_2) \in SF \wedge (n_1 \in (E_s \cup G^s_+ \cup G^j_+) \vee \\
& \quad (n_1 \in \mathcal{T} \wedge obj(n_1) = \emptyset))\} \cup \\
& \{(t_{(i,c,s)}, p_{(i,n)}) \mid (i,n) \in SF \wedge i \in \mathcal{T} \wedge (c,s) \in obj(i)\} \cup \\
& \{(p_{(n_1,g)}, t_{(g,n_2)}) \mid (n_1,g) \in SF \wedge (g,n_2) \in SF \wedge g \in G^s_\times\} \cup \\
& \{(t_{(g,n)}, p_{(g,n)}) \mid (g,n) \in SF \wedge g \in G^s_\times\} \cup \\
& \{(p_{(n,g)}, t_{(n,g)}) \mid (n,g) \in SF \wedge g \in G^j_\times\} \cup \\
& \{(t_{(n_1,g)}, p_{(g,n_2)}) \mid (n_1,g) \in SF \wedge (g,n_2) \in SF \wedge g \in G^j_\times\} \cup \\
& \{(p_{(n_1,g)}, t_{n_2}) \mid (n_1,g) \in SF \wedge (g,n_2) \in SF \wedge g \in G^s_e\}
\end{aligned}
$$

In the following, the mapping is extended with message places to incorporate data semantics. For each class in the shared data model, an initialization place P_{MI} is added to ensure that only one instance of a message object is created during conversation. In addition, message places are introduced for each class and state combination for each participant:

$$
\begin{aligned}
P_{MI} &= \{p_c \mid c \in C\} \\
P_M &= \{p_{(r,c,s)} \mid r \in R \wedge c \in C \wedge s \in S_c\}
\end{aligned}
$$

Silent transitions are introduced to create and modify message objects and their state. The creation of message objects may be subject to constraints due to



(a) One-to-One (b) Many-to-One

Fig. 6. Creation of message objects in data-enhanced interaction Petri nets considering different multiplicity constraints.

relations as specified in Sect. 4.1. Hence, the mapping enforces these constraints by following the rules depicted in Fig. 6. The rules ensure that for one-to-one relations, the message objects are created at the same time, and for many-to-one relations, the related message object must exist for creation, as illustrated in Fig. 5 for the creation of an 'Import Permit' message object. For this purpose, two auxiliary functions are introduced. $rel_{(1,1)} : C \to 2^C$ associates each class with a set of classes that have one-to-one relations to the given class, including the given class itself, while also considering transitive relations. Correspondingly, we define a function $rel_{(n,1)} : C \to 2^C$ which returns a set of classes having a many-to-one relation to the given class, so that $\forall c_1, c_2 \in C : c_1 \in rel_{(n,1)}(c_2) \iff mult((c_1, c_2)) = (n, 1)$.

Classes with a one-to-many relation to the given class may have multiple states. Hence, multiple transitions are required to create a message object. Given a class $c \in C$, we define the set of all class and state combinations having a many-to-one relation to c or to a class having a one-to-one relation with c as follows:

$$CS_c^{(n,1)} = \{(c', s) \mid \exists c'' \in rel_{(1,1)}(c) : c' \in rel_{(n,1)}(c'') \wedge s \in S_{c'}\}$$

Since a class can only be in a single state for a participant, we define the set of relation dependencies including sets of all possible combinations of classes and states having a corresponding many-to-one relation to c as follows:

$$RD_c = \{CS \mid CS \subseteq CS_c^{(n,1)} \wedge \forall (c', s') \in CS_c^{(n,1)} : \exists! s \in S_{c'} : (c', s) \in CS\}$$

Hence, the manipulation of message objects is represented by a set of silent message modification transitions T_M:

$$T_M = \{t_{(r,C',RD,s)} \mid c \in C \wedge C' = rel_{(1,1)}(c) \wedge RD \in RD_c \wedge s \in S_c^i \wedge r = role(s)\} \cup \{t_{(r,c,s_1,s_2)} \mid c \in C \wedge (s_1, s_2) \in \delta_c \wedge r = role((s_1, s_2))\}$$

The additional nodes need to be connected with arcs to represent the respective creation and state change behaviors. The set of message flow arcs F_M can

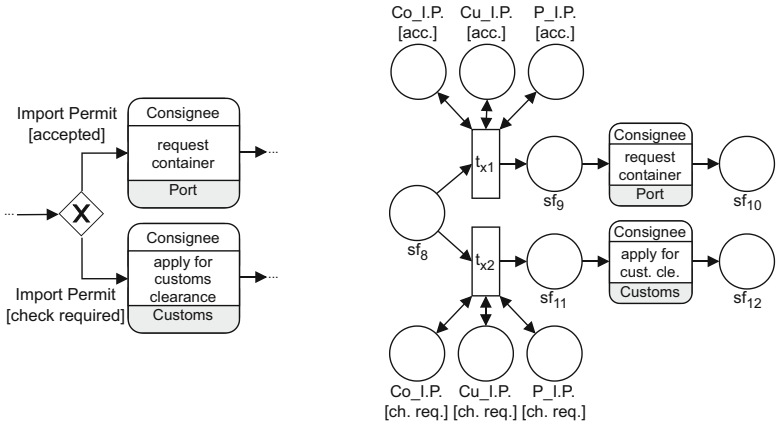
be partitioned into disjoint sets of message object manipulation arcs F_M^O, interaction dependency arcs F_M^I, and exclusive gateway dependency arcs $F_M^\times$. The set of message object manipulation arcs F_M^O connects message places P_M and initialization places P_{MI} with message modification transitions T_M according to the shared data model and distributed object lifecycles:

$$
\begin{aligned}
F_M^O = {} & \{(t_e, p_c) \mid e \in E_s \wedge c \in C\} \cup \\
& \{(p_c, t_{(r,C',RD,s)}) \mid c \in C \wedge C' = rel_{(1,1)}(c) \wedge RD \in RD_c \wedge s \in S_c^i \\
& \wedge r = role(s)\} \cup \\
& \{(t_{(r,C',RD,s)}, p_{(r,c,s)}) \mid c \in C \wedge C' = rel_{(1,1)}(c) \wedge RD \in RD_c \wedge s \in S_c^i \wedge \\
& r = role(s)\} \cup \\
& \{(p_{(r,c,s)}, t_{(r,C',RD,s')}) \mid RD \in RD_c \wedge c, c' \in C \wedge C' = rel_{(1,1)}(c') \wedge c \notin C' \wedge \\
& (c,s) \in RD \wedge s' \in S_{c'}^i \wedge r = role(s')\} \cup \\
& \{(t_{(r,C',RD,s')}, p_{(r,c,s)}) \mid RD \in RC_c \wedge c, c' \in C \wedge C' = rel_{(1,1)}(c') \wedge c \notin C' \wedge \\
& (c,s) \in RD \wedge s' \in S_{c'}^i \wedge r = role(s')\} \cup \\
& \{(p_{(r,c,s_1)}, t_{(r,c,s_1,s_2)}) \mid c \in C \wedge (s_1, s_2) \in \delta_c \wedge r = role((s_1, s_2))\} \cup \\
& \{(t_{(r,c,s_1,s_2)}, p_{(r,c,s_2)}) \mid c \in C \wedge (s_1, s_2) \in \delta_c \wedge r = role((s_1, s_2))\}
\end{aligned}
$$

Furthermore, message places are associated with interaction transitions to represent the data transfer. For this purpose, each interaction transition must read (i.e., consume and produce) the token from a message place of the task initiator with the appropriate class and state to ensure that the initiator is aware of the data to be sent as defined in F_M^I. Thus, if a message sender is unaware of the required data during a conversation, the transition cannot fire. In the other case, when the transition is fired, a token is produced in the appropriate message place of the receiver. In addition, reset arcs RF are added to reset any message place of the same class as the exchanged message object on the receiver side to prevent participants from having message objects of the same class in multiple states, as depicted in Fig. 4 and Fig. 5.

$$
\begin{aligned}
F_M^I = {} & \{(p_{(r,c,s)}, t_{(i,c,s)}) \mid i \in \mathcal{T} \wedge r = init(i) \ \wedge (c,s) \in obj(i)\} \cup \\
& \{(t_{(i,c,s)}, p_{(r,c,s)}) \mid i \in \mathcal{T} \wedge \ r \in \{init(i), resp(i)\} \wedge (c,s) \in obj(i)\} \\
RF = {} & \{(p_{(r,c,s)}, t_{(i,c,s')}) \mid i \in \mathcal{T} \wedge r = resp(i) \wedge (c,s') \in obj(i) \wedge s \in S_c\}
\end{aligned}
$$

Finally, according to the semantics of exclusive gateways, it must be ensured that all affected participants have the same view of the data on which the decision is based [15]. To enforce semantics in a data-enhanced IPN, silent transitions representing the decision are required to read the appropriate message places of all affected participants, as illustrated in Fig. 7. Inconsistencies in the participants' data would result in transitions associated with the gateway not being able to fire, thus ensuring data consistency for a firing sequence.



(a) Choreography diagram (b) Data-enhanced interaction Petri net

Fig. 7. Exclusive gateway deciding on subsequent path based on state of the import permit (I.P.) represented as choreography diagram (a) and data-enhanced interaction Petri net (b).

To narrow down the affected participants for a decision, we refer to the concept of *single-entry single-exit* (SESE) regions [8]. A SESE region includes all elements on a path between an exclusive gateway split and an exclusive gateway join or end event, with all paths originating from the split leading to either a join or end events. In the case of a loop, all paths but one may lead back to the initial split. Hence, given the example in Fig. 1, since not all paths originating from the exclusive gateway split lead to an exclusive gateway join, the SESE region extends until the end of the choreography. We argue that all participants involved in tasks in a SESE region started by an exclusive split can be considered affected, since execution within the region depends on the initial decision. The behavior after the SESE region is independent of the decision and can be neglected. Therefore, we define the auxiliary function $sese_{\times} : G^s_{\times} \rightarrow 2^R$, which returns the set of participants involved in tasks in a SESE region started by a given exclusive split.

$$F^{\times}_M = \{(p_{(r,c,s)}, t_{(g,n)}) \mid g \in G^s_{\times} \wedge (g,n) \in SF \wedge r \in sese_{\times}(g) \wedge (c,s) = grd((g,n))\} \cup$$
$$\{(t_{(g,n)}, p_{(r,c,s)}) \mid g \in G^s_{\times} \wedge (g,n) \in SF \wedge r \in sese_{\times}(g) \wedge (c,s) = grd((g,n))\}$$

Consequently, the mapping inherently enforces that each sender must be aware of the message objects to be sent and that decisions can only be made if the affected participants are aware of the corresponding message object in a consistent state. In addition, dependencies within and between message objects are reflected according to the specifications of the shared data model and distributed object lifecycles. Thus, the approach provides a foundation for specifying and analyzing data exchange in process choreographies.

5 Discussion

In the following, the limitations of the presented extension are discussed. First, message objects and states provide only an abstract view of the exchanged data to focus on the aspects relevant to the business case. While this allows a conceptual design of the data exchange, the mapping between states and actual attribute values requires further research. Furthermore, while the supplementary models allow the definition of dependencies for data specified in choreography diagrams, the approach relies on the consistency between all involved models. Therefore, the development of techniques to automatically verify the consistency between these models can facilitate the design. In addition, the extension does not support multi-instance tasks, messages, or participants. The impact of multi-instance behavior on the semantics of data exchange remains to be explored. Although intermediate events are not supported, their inclusion requires only minor extensions, which are not considered due to space limitations.

The presented extension serves as a foundation for the analysis of data exchange in choreographies. Although the detection and classification of data exchange errors is beyond the scope of this work, potential errors may already be revealed by detecting deadlocks in the possible firing sequences [2]. Since the execution semantics require the initiator of a task to be aware of the message object to be sent, data awareness can be addressed in this way. Correspondingly, exclusive gateway splits can only be executed if the participants have a consistent view of the data, which addresses the data consistency concern. However, since message places may still contain tokens after a conversation terminated, a classical soundness analysis is not applicable to the presented approach [1]. Nevertheless, the extension provides a more concise specification of the data exchange in choreographies, allowing a more in-depth analysis of the message flow.

6 Related Work

While formal execution semantics for interaction models already exist in the literature, most work focuses on the ordering of interactions. Decker et al. defines the execution semantics for the interaction-centric modeling language Let's Dance using π-calculus [6]. In addition, interaction Petri nets, introduced in [5], describe the behavior of iBPMN choreographies. Najem et al. provide a mapping of choreography diagrams to colored Petri nets, which allows detecting deadlocks in the control flow of choreographies [13]. Furthermore, Corradini et al. use a Backus Normal Form syntax to check the conformance between BPMN 2.0 choreography and collaboration diagrams [3]. However, these works neglect the role of data. Based on interaction Petri nets, our work aims to extend the execution semantics of choreographies with a data perspective.

The data exchange between processes in a choreography is investigated by Meyer et al. [12]. The authors introduce a model-driven approach to enable an automated data exchange in choreographies using a global data model. While

a mapping between local and global data allows reasoning about data dependencies, data awareness is not addressed. Knuplesch et al. extend the notion of BPMN choreography diagrams with virtual data objects as variables for routing conditions [10]. The authors combine interaction Petri nets and workflow nets with data. However, data dependencies are not considered. Furthermore, Nikaj et al. propose a RESTful representation for choreography diagrams [14]. Since participants must be aware of URLs to invoke them, the approach takes data awareness into account. Nevertheless, data dependencies are neglected.

The decoupling of message and data flow is discussed by Hahn et al. [9]. The authors introduce a middleware to coordinate the propagation of changes to shared data objects used in the local processes of collaborating participants. In contrast to our work, the approach relies on interconnection models and requires insight into the local data flow of the participants. Finally, Köpke et al. propose an approach to model the data flow of interorganizational process models starting from a global process model assuming a central data store [11]. In a second step, the data flow is then distributed among the participants. Due to the initial holistic view, the correctness of the data flow can be ensured with existing techniques. However, unlike our work, the approach requires detailed insight into the organization-internal process behavior of participants, which may complicate collaboration with untrusted organizations.

7 Conclusion

This paper proposes a novel way to describe data exchange in BPMN choreography diagrams by using a shared data model and distributed object lifecycles as supplemental models to define data relations and behavior. In addition, we extended interaction Petri nets with a data perspective to provide a formal basis for defining the execution semantics of data exchange in choreographies. Finally, a mapping of supplemented choreography diagrams to data-enhanced interaction Petri nets is provided, allowing the analysis of the data flow of choreography diagrams. The approach addresses the need for more concise semantics for data exchange to identify data-related flaws in interaction behavior at design time.

For future research, we plan to develop tools to facilitate the modeling and analysis of data exchange in choreographies, as well as a method for verifying the consistency of local and global data flow given a local process model. In addition, we plan to extend our mapping to support multi-instance behavior and more complex relations among message objects, and aim to uncover patterns and antipatterns in interorganizational data exchange. Despite the potential extensions, our proposal already allows for a more precise specification of data exchange for choreographies.

Acknowledgement. Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 450612067

References

1. van der Aalst, W.M.P., et al.: Soundness of workflow nets: classification, decidability, and analysis. Formal Aspects Comput. **23**(3), 333–363 (2011). https://doi.org/10.1007/s00165-010-0161-4
2. Chu, F., Xie, X.: Deadlock analysis of Petri nets using siphons and mathematical programming. IEEE Trans. Robot. Autom. **13**(6), 793–804 (1997). https://doi.org/10.1109/70.650158
3. Corradini, F., Morichetta, A., Polini, A., Re, B., Tiezzi, F.: Collaboration vs. choreography conformance in BPMN. Log. Methods Comput. Sci. **16**(4) (2020)
4. Decker, G., Weske, M.: Local enforceability in interaction petri nets. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 305–319. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_22
5. Decker, G., Weske, M.: Interaction-centric modeling of process choreographies. Inf. Syst. **36**(2), 292–312 (2011). https://doi.org/10.1016/j.is.2010.06.005
6. Decker, G., Zaha, J.M., Dumas, M.: Execution semantics for service choreographies. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 163–177. Springer, Heidelberg (2006). https://doi.org/10.1007/11841197_11
7. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. **50**(12), 1281–1294 (2008). https://doi.org/10.1016/j.infsof.2008.02.006
8. Dumas, M., García-Bañuelos, L., Polyvyanyy, A.: Unraveling unstructured process models. In: Mendling, J., Weidlich, M., Weske, M. (eds.) BPMN 2010. LNBIP, vol. 67, pp. 1–7. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-16298-5_1
9. Hahn, M., Breitenbücher, U., Kopp, O., Leymann, F.: Modeling and execution of data-aware choreographies: an overview. Comput. Sci. Res. Dev. **33**(3–4), 329–340 (2018). https://doi.org/10.1007/s00450-017-0387-y
10. Knuplesch, D., Pryss, R., Reichert, M.: Data-aware interaction in distributed and collaborative workflows: modeling, semantics, correctness. In: CollaborateCom 2012, Pittsburgh, PA, USA, October 14–17, 2012, pp. 223–232. ICST/IEEE (2012). https://doi.org/10.4108/icst.collaboratecom.2012.250443
11. Köpke, J., Franceschetti, M., Eder, J.: Optimizing data-flow implementations for inter-organizational processes. Distrib. Parallel Databases **37**(4), 651–695 (2019). https://doi.org/10.1007/s10619-018-7251-3
12. Meyer, A., Pufahl, L., Batoulis, K., Fahland, D., Weske, M.: Automating data exchange in process choreographies. Inf. Syst. **53**, 296–329 (2015). https://doi.org/10.1016/j.is.2015.03.008
13. Najem, T., Perucci, A.: Mapping BPMN2 service choreographies to colored petri nets. In: Camara, J., Steffen, M. (eds.) SEFM 2019. LNCS, vol. 12226, pp. 85–100. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-57506-9_8
14. Nikaj, A., Weske, M.: Formal Specification of RESTful choreography properties. In: Bozzon, A., Cudre-Maroux, P., Pautasso, C. (eds.) ICWE 2016. LNCS, vol. 9671, pp. 365–372. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-38791-8_21
15. Object Management Group (OMG): business process model and notation (BPMN), Version 2.0.2 (2014). https://www.omg.org/spec/BPMN/2.0.2/