

# Data Consistency as a Criterion for Process Choreography Design\*

Tom Lichtenstein, Mathias Weske

Hasso Plattner Institute, University of Potsdam, Potsdam, Germany

## Abstract

Process choreographies involve the exchange of data and goods between different business actors. Since inconsistencies in data shared among choreography participants can lead to conflicts, the preservation of data consistency needs to be considered during choreography design. However, current choreography modeling languages lack data modeling capabilities. As a result, potential conflicts due to data inconsistencies may remain undetected during design time. To address the lack of information, this paper proposes a modeling framework enabling the coordination of interorganizational data management in process choreographies with respect to data consistency. The framework enriches the BPMN standard with data consistency-related specifications at the public and private process level, and provides rules for validating the compatibility of the specifications at both levels. Furthermore, the framework is applied to an exemplary ticket reservation choreography, demonstrating the framework's capabilities for detecting conflicts due to data inconsistencies at design time.

## Keywords

Process Choreography, Data Consistency, Design Time

## 1. Introduction

In today's interconnected economy, businesses and organizations increasingly collaborate in order to achieve their goals. Therefore, current business processes typically involve the exchange of data and goods with business processes of other organizations, resulting in *process choreographies* [1]. Since the data exchanged may affect the individual process behavior of a choreography participant, *data consistency*, i.e., a uniform view of the data shared with multiple participants [2], is a desirable property to maintain during choreography execution [3]. Local updates to data shared among choreography participants may lead to inconsistent views of the data, which can ultimately lead to conflicting interaction behavior [4]. Since conflicts affect operations and incur costs, the interaction behavior of choreographies must be designed carefully [1, 3, 5]. Modeling languages such as *Business Process Model and Notation* (BPMN) [6] can support the design of process choreographies. However, current choreography modeling languages only have limited data modeling capabilities. As a result, conflicts resulting from data inconsistencies at runtime may remain undetected at design time.

---

ER'2022 Forum and Symposium, October 17-20, 2022, Online

\* Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 450612067

✉ tom.lichtenstein@hpi.de (T. Lichtenstein); mathias.weske@hpi.de (M. Weske)

🆔 0000-0001-5585-1003 (T. Lichtenstein); 0000-0002-3346-2442 (M. Weske)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

This paper aims to address the lack of information in the design of process choreographies by introducing a modeling framework that enables the coordination of consistent data management across organizational boundaries without requiring individual participants to reveal details about their internal process logic. The framework extends the BPMN modeling standard by integrating data consistency-related information into the modeling of public and private processes. Furthermore, rules are introduced to ensure the compatibility of the private data flow with the consistency specifications of the public process, also considering concurrent data access by other process instances of a single choreography participant.

The remainder of this paper is organized as follows: Section 2 briefly introduces the underlying concepts relevant to the framework and a motivating example. Next, in Section 3, the modeling framework is proposed, which is the main contribution of the paper. Section 4 then discusses the applicability of the approach, followed by an overview of related work in Section 5. Finally, Section 6 summarizes the findings of this work.

## 2. Preliminaries and Motivating Example

This paper presents a novel concept for integrating data consistency information into the design of process choreographies. As a foundation, the concepts of data consistency, data modeling in BPMN, and process choreographies are explained in this section. In addition, a motivating example is presented to illustrate the relevance of considering data consistency during process choreography design.

### 2.1. Data Consistency

In the data management literature, the definition of data consistency mainly refers to the ACID transaction model for database management systems [7]. The CAP theorem transfers the definition to distributed systems [8]. Since process choreographies can be classified as distributed systems, this work refers to the definition of the CAP theorem. According to the theorem, a distributed system is consistent if all participating nodes return the same data for the same request, i.e., all nodes have the same view of the data.

Since maintaining total consistency among all nodes implies overhead, consistency models are introduced to relax the degree of consistency in a system to enable more efficient operation. A consistency model represents a contract between processes and a (distributed) data store, where the data store guarantees consistent behavior if the processes follow the rules specified by the consistency model [2]. While strong consistency models require consistency at all times, weaker consistency models, such as eventual consistency, allow for partial inconsistencies that converge to a consistent state over time. Nevertheless, the degree of consistency required by a system depends strongly on its application.

### 2.2. Data Modeling in BPMN

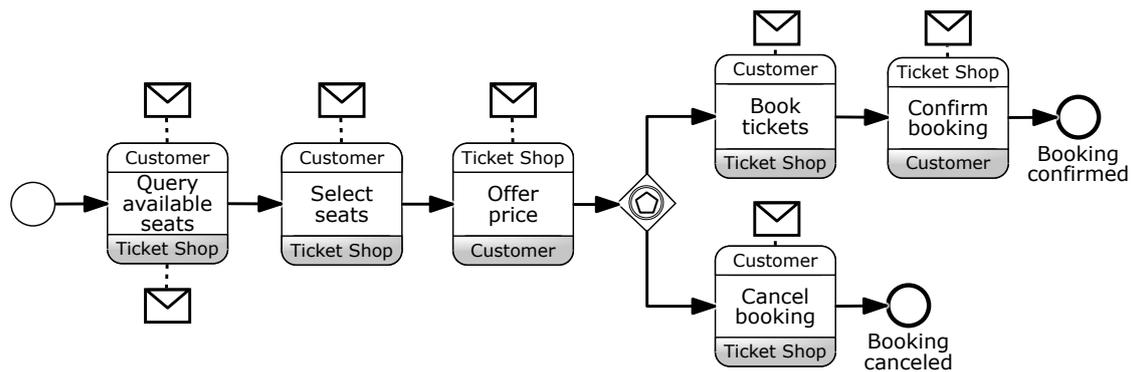
In BPMN, data used during process execution is represented by *data objects* [6]. A data object refers to information required or produced by the execution of an activity (i.e., operational data [4]). Although data objects can refer to physical artifacts such as paper documents or

products [1], this paper limits the scope of data objects to only referring to data that can be captured by information systems. Data objects can be created, modified, or accessed using data operations performed by activities [9]. From an implementation perspective, each data operation can be viewed as a *read* or *write* operation [3]. Following the BPMN standard, read operations are represented as incoming data associations, while write operations are represented as outgoing data associations from the perspective of an activity. The data flow of a process thus results from the data operations performed by each activity. In the following, it is assumed that for each activity, the read operations are always performed before the write operations. Inspired by [9], the following subset of BPMN is used to define data objects in process models:

**Definition 1.** (*Process Model*) A process model is considered to be a tuple  $P = (N, D, DS, CF, DF, S)$ , where  $N \subseteq A \cup G \cup E$  is a finite, non-empty set of nodes containing activities  $A$ , gateways  $G$ , and events  $E$ ,  $D$  is a finite set of data objects, and  $DS$  is a finite set of data stores. Furthermore,  $CF \subseteq N \times N$  expresses the control flow relation, and  $DF \subseteq (A \times D) \cup (D \times A)$  captures the data flow relation with the first element in each tuple indicating the origin and the second element specifying the target of a data flow association. Finally,  $S \subseteq D \times DS$  determines the data persistence relation.

### 2.3. Process Choreography

Process choreographies describe the interaction behavior between process orchestrations of two or more collaborating business actors, e.g., enterprises, customers, organizations [1]. The interactions are limited to the exchange of messages. To represent process choreographies, BPMN 2.0 [6] introduces *choreography diagrams* as a standard for choreography modeling. Compared to BPMN collaboration diagrams, choreography diagrams abstract from the internal process behavior (i.e., *private process*) and focus only on the interactions (i.e., *public process*). A choreography diagram consists of a set of choreography tasks, each representing a message exchange between two or more participants, and an optional response. Similar to BPMN process models, choreography tasks follow a causal order determined by sequence flows. Decisions and parallel behavior among tasks can be expressed by gateways. Nevertheless, choreography



**Figure 1:** Choreography diagram describing a ticket reservation choreography between a customer and an online ticket shop

diagrams do not specify the content or structure of the exchanged messages. An example of a choreography diagram is depicted in Figure 1.

## 2.4. Motivating Example

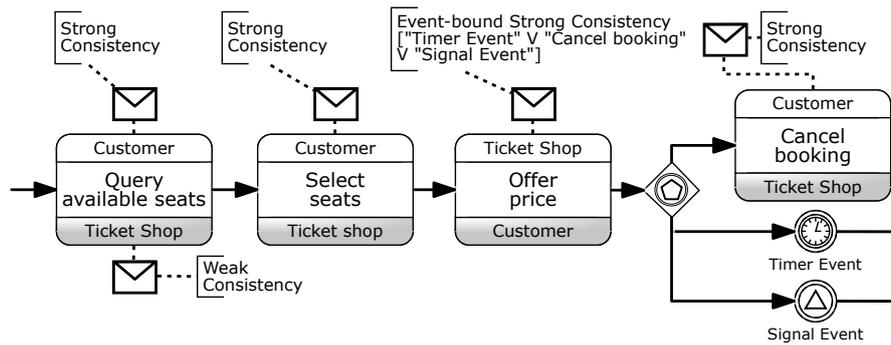
In the following, the need for considering data consistency during process choreography design is illustrated by an exemplary ticket reservation choreography shown in Figure 1. The choreography starts with the customer querying and selecting available seats. Based on the price offer, the customer decides whether to cancel the booking or book the ticket, which is then confirmed by the ticket shop. Considering a concurrent execution of the choreography, conflicts may occur due to data inconsistencies. In case of two customers concurrently selecting overlapping seats, as soon as the first customer's booking is confirmed, the second customer has an inconsistent view on the available seats. Moreover, considering that each seat can be sold to one customer, if the second customer decides to book the tickets, this results in a conflict which requires additional interaction behavior for compensation. However, the potential need for compensation is not evident by the choreography diagram.

## 3. Data Consistency-aware Process Choreography Design

To address the lack of information regarding data consistency during process choreography design, this section introduces a modeling framework to coordinate the interorganizational data management of process choreographies. The framework extends BPMN choreography and collaboration diagrams with data consistency-specific annotations on the public and private process level. In Section 3.1, *consistency constraints* are introduced to choreography diagrams to enable the specification of data consistency information on the public process. Next, Section 3.2 describes an extension to private data flow modeling that adds specifications for data accessibility. In Section 3.3, rules are defined based on the extensions to ensure the compatibility of data accessibility specifications of the private process with the consistency constraints of the public process. As a result, data consistency information can be communicated at the public process level and verified at the private process level.

### 3.1. Consistency Constraints

While BPMN choreography diagrams allow collaborating organizations to align their business processes without revealing internal process details, they lack information about the individual management of the exchanged data. As a result, the ambiguity of data consistency during design time can lead to conflicts at runtime. This section introduces the concept of *consistency constraints* to enable the specification of data consistency information in process choreography diagrams. Similar to consistency models that specify rules to ensure data consistency in a distributed system, consistency constraints determine the degree of consistency that is guaranteed for data exchanged between participants. However, unlike consistency models, the scope of a consistency constraint is limited to data exchanged by a single message. Restricting the scope of the constraints to messages allows for more flexible data management, as each interaction can be tailored to its individual consistency needs.



**Figure 2:** Consistency constraints specified on an extended excerpt of the ticket reservation choreography

As illustrated in Figure 2, the consistency constraints are integrated into choreography diagrams via annotations on the respective message elements. Specifying the constraints on the message elements instead on the choreography tasks allows multiple messages associated with the same choreography task to have different consistency constraints. By using annotations instead of a novel graphical notion, the extended model still complies to the BPMN 2.0 standard [6]. In the following, three consistency constraints *strong consistency*, *weak consistency*, and *event-bound strong consistency* are introduced.

### 3.1.1. Strong Consistency

In terms of maintaining a consistent view between participants, strong consistency is the most restrictive, but also the most reliable consistency constraint. Strong consistency guarantees that all participants always have the same view of the shared data. Consequently, any changes to the data must be immediately propagated to each corresponding participant to ensure a consistent view. However, since the exchanged data may have already been used by other participants as a basis for decisions and other data operations, modifying strongly consistent data could still lead to inconsistencies. Therefore, data that is exchanged with strong consistency is considered immutable and cannot be changed after it is sent. This restriction may also affect concurrent process instances accessing the exchanged data. A special case for strongly consistent interactions is the resending of a message when the interaction is looped. Since the data of each loop iteration may depend on a local scope, e.g., each loop iteration relates to a single ticket purchase, the data to be exchanged by the same task may change for each iteration. However, due to strong consistency, each received data object within this scope must be treated independently and does not overwrite data objects originating from previous messages.

Since strong consistency guarantees that the exchanged data remains unchanged, the receiver can safely integrate the data into local data operations and decisions. Thus, in the example depicted in Figure 2, the ticket shop is assured that the selected seats are consistent during the remaining execution of the choreography. Since inconsistencies cannot occur under strong consistency, compensation behavior is not required. Nevertheless, due to the inherent restrictions regarding the modification of exchanged data, this constraint may affect concurrent process instances that require access to the corresponding data. Therefore, strong consistency is most

suitable for exchanged data that is either rarely accessed by concurrent processes or infrequently modified.

### **3.1.2. Weak Consistency**

In contrast to strong consistency, weak consistency allows for more relaxed consistency between participants, as it does not restrict the accessibility of exchanged data. Therefore, exchanged data can also be modified from outside the scope of choreography execution (e.g., by concurrent process instances). Considering the example choreography, if the ticket shop exchanges data about available seats with weak consistency, concurrent process instances can still modify the availability of specific seats, which may lead to inconsistent views. As a result, the receiver has no guarantee on the consistency of the exchanged data, which must be taken into account when using the data during process execution. Since possible inconsistencies can lead to conflicts (e.g., by violating local integrity constraints), weak consistency requires the incorporation of compensation behavior to synchronize the divergent views of the data. In addition, if the interaction is looped, the received data may update or overwrite previously received data if it contains more recent values. Given that the compensation behavior can affect the flow of a conversation, weak consistency should be applied to messages exchanging data that is modified frequently by concurrent process instances, whereas the changes rarely lead to conflicts.

### **3.1.3. Event-bound Strong Consistency**

While strong consistency can effectively avoid inconsistencies, restricting write access to shared data for the remaining choreography is not viable in certain scenarios. In addition, data consistency may only be required within a specific part of a choreography. For example, a consistent view of available seats or the price of a ticket is no longer required if the customer has cancelled the booking. To address this shortcoming of strong consistency, event-bound strong consistency combines concepts of strong consistency and weak consistency to provide consistency within a predefined part of a choreography. To this end, event-bound strong consistency requires the specification of one or more events (also including message-related events which are represented as choreography tasks in choreography diagrams) upon whose occurrence consistency is no longer mandatory. The corresponding events are referenced by their names in the constraint specification, as depicted in Figure 2. All specified events need to be included in the choreography diagram and must occur after the sending of the message. It should be noted that the events do not have to directly follow the sending task, but can occur at any point in the choreography after the interaction. The specification of multiple events follows disjunctive semantics. Using this information, the event-bound strong consistency first enforces the strong consistency constraint properties on the exchanged data until one of the specified events occurs in the conversation. After the occurrence, the shared data follows the properties of weak consistency, thus re-enabling data modifications.

An important aspect of event-bound strong consistency is the selection of events that modify the consistency properties, since all participants sharing the data must be able to capture each event simultaneously. The framework focuses on message, timer, and signal events as they can be applied in a distributed environment and are supported by choreography diagrams. Message

and signal events must be invoked by the participant that originally shared the data with event-bound strong consistency. Timer events must either be specified in a location-independent format or determine a time frame after the preceding interaction to avoid ambiguity.

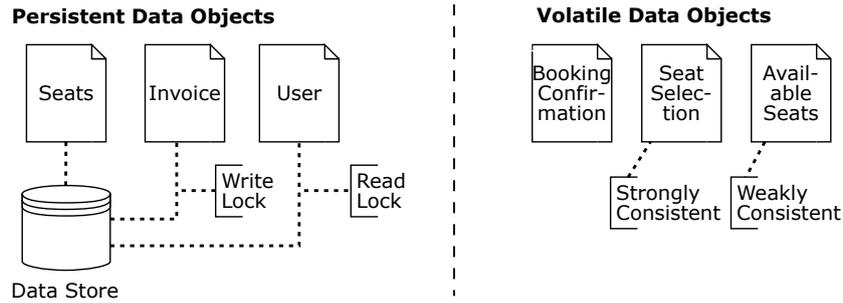
### 3.2. Data Object Accessibility Specifications

In order to ensure the compatibility of the private process's data management with the consistency specifications, the accessibility of data objects must be considered during design. Therefore, this section introduces accessibility specifications for data objects that also consider shared access with concurrent process instances and choreography participants. The framework distinguishes between two types of data objects: *persistent data objects* and *volatile data objects*. Both types differ in their visibility to concurrent process instances and therefore require different access specifications. The data object types are explained in more detail below.

#### 3.2.1. Persistent Data Objects

Data objects that are stored in a data store are referred to as persistent data objects. Thus, according to Definition 1, the set of persistent data objects in a process model is defined as  $PD = \{d \in D \mid \exists ds \in DS : (d, ds) \in S\}$ . To visually distinguish persistent data objects from volatile data objects, their representation maintains an association with a data store reference, as illustrated in Figure 3. Since persistent data objects are managed in central data stores, it is assumed that they can also be accessed by concurrent process instances of the same participant. In order to operate consistently on data objects that may be accessed by multiple process instances, concurrent access to persistent data objects needs to be managed by a concurrency control mechanisms. In this work, we focus on locking mechanisms for concurrency control, as they are well understood for concurrent systems [2, 7]. If a process instance acquires a lock on a persistent data object, the locking mechanism prevents concurrent process instances from modifying the data object. A locking mechanism is particularly relevant to enable strongly consistent data exchanges, as it protects the data from manipulations and avoids rollbacks that would require additional compensation behavior.

The framework introduces two types of locks for persistent data objects: *read locks* and *write locks*. While read locks only allow read access and thus do not permit write access to the data object, write locks enable full access for the process instance holding the lock. The locking mechanism should follow the concurrent reads and exclusive writes memory model [2]. According to this model, multiple process instances can acquire read locks for the same data object simultaneously, while a write lock can only be acquired by one process instance at a time. Therefore, given the set of process instances  $PI$  and the set of persistent data objects  $PD$ , the set of process instances holding a read lock on a persistent data object is determined by  $L_r : PD \rightarrow \mathcal{P}(PI)$ . Correspondingly,  $L_w : PD \rightarrow PI \cup \{\emptyset\}$  specifies for each persistent data object the process instance that holds a write lock for the object, while  $\emptyset$  denotes that no instance is holding a write lock for the particular object. To obtain a write lock, no process instance may hold a lock on the persistent data object, including read locks:  $\forall d \in PD : L_w(d) \neq \emptyset \rightarrow L_r(d) = \emptyset$ . In turn, no process instance may hold a write lock on the data object to obtain a read lock:  $\forall d \in PD : L_r(d) \neq \emptyset \rightarrow L_w(d) = \emptyset$ . If a lock cannot be



**Figure 3:** Data object accessibility specifications for persistent and volatile data objects

obtained by a process instance due to the aforementioned conditions, the activity accessing the data object must wait until the corresponding lock has been released. Therefore, the locking mechanism introduces blocking behavior. However, by distinguishing between read and write locks, data management can be tailored to meet the requirements of consistency constraints while still allowing concurrent data access in certain scenarios. The lock type required for a persistent data object in a process execution is specified as an annotation attached to the association between the data object and the data store, as shown in Figure 3.

Furthermore, the framework also supports *unlocked* access to data objects, which is indicated by the absence of a lock annotation on the association. Accessing a persistent data object without a lock enables optimistic concurrency control strategies that allow concurrent reads and writes by multiple process instances, potentially increasing the throughput. While read operations are always allowed for unlocked persistent data objects, write operations can only be performed if no other process instance holds a lock on the data object. Nevertheless, concurrent manipulation of the data objects can cause conflicts that must be compensated for in the execution of a process instance (e.g., by violating integrity constraints). Therefore, each activity writing to an unlocked persistent data objects requires an error boundary event, providing exceptional behavior in case of a conflict.

### 3.2.2. Volatile Data Objects

In contrast to persistent data objects, volatile data objects only exist within the scope of a single process instance. Hence, volatile data objects are not accessible to concurrent process instances. Given a process model, the set of volatile data objects is defined as  $VD = \{d \in D \mid \forall ds \in DS : (d, ds) \notin S\}$ . Due to their isolation from concurrent process instances, volatile data objects are generally unconstrained in their data operations, whereas data operations by concurrent execution paths within a single process must be handled by the process engine. However, volatile data objects may be part of messages exchanged between participants. Since consistency constraints can impose accessibility constraints on data objects, the framework includes two additional annotations *strongly consistent* and *weakly consistent* for volatile data objects, as depicted in Figure 3. Data objects associated with either annotation are restricted to read access. The difference in semantics between the two annotations is discussed in the next section.

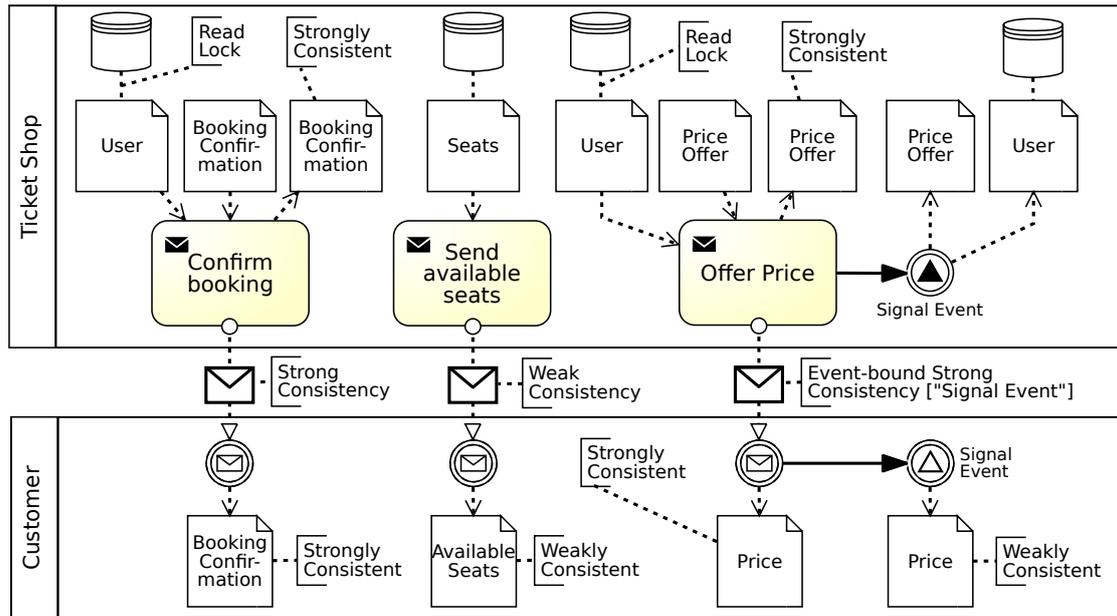
### 3.3. Data Flow Compatibility Rules

The impact of consistency constraints on the local management of exchanged data is twofold. On the one hand, consistency constraints impose data management restrictions on the sender's side, as they may require the sender to protect exchanged data from modification. On the other hand, they provide information about the degree of consistency that can be expected by the recipient. This section provides rules to ensure the compatibility between the public consistency constraints and the private data flow using the language extensions introduced in previous sections.

The rules are based on the assumption that the data flow implies dependencies between data objects. Thus, if an activity reads a data object and writes to another data object, the data written is expected to depend on the outcome of the read operation. As a result, the written data object inherits consistency properties of the read data object. For example, if the ticket shop creates a volatile data object 'Available Seats' based on an unlocked persistent data object 'Seats', the content of the volatile data object is still subject to inconsistencies due to its dependency on the 'Seats' data object. This dependency also applies transitively. Accordingly, the content of a message is expected to depend on the data objects read by the sending activity. Hence, given the data objects  $d_1, \dots, d_i \in D$  read by the sending activity, a message  $m \in M$  is considered a tuple  $m = (name, c, v(d_1, \dots, d_i))$ , where *name* captures a textual description of the interaction, *c* determines the consistency constraint assigned to the message, and  $v : \mathcal{P}(D) \rightarrow \mathcal{P}(D)$  specifies a view on the data objects representing the content of the message. On the receiving end, the content can be interpreted as a different set of data objects. Furthermore, it is assumed that received data is only accessible to the process instance receiving the data. An overview of the data flow compatibility rules is provided in Figure 4 using three interactions from the ticket reservation choreography as examples, each of which is associated with a different consistency constraint. The implications on the private data flow of the participants are described in more detail below.

Since strong consistency ensures immutability of exchanged data, it implies strict rules on the private data flow. As a message's content depends on the data objects read by the sending activity, those data objects need to be protected from modifications. In the case of persistent data objects, read locks are required to be held until the end of the choreography execution to avoid modifications by concurrent process instances and by the sending process instance. Volatile data objects, on the other hand, cannot be accessed by concurrent process instances. However, if their content depends on persistent data objects according to the data flow, these must be protected by a read lock. To protect volatile data objects from changes, they become strongly consistent after being sent, as shown in Figure 4. While the change in consistency is denoted by a write operation, the content of the data object itself is not modified in this case. At the receiving end, a message with strong consistency can produce one or multiple strongly consistent data objects. Since strongly consistent data objects are immutable for both participants, they can be used safely in local data operations and decisions.

Weak consistency lowers the restrictions on the sender's side. Therefore, any data object may serve as the content of a message, as the constraint does not imply restrictions on the data objects. As a consequence, the recipient can only derive one or multiple weakly consistent data objects from a corresponding message. While weakly consistent data objects share the same



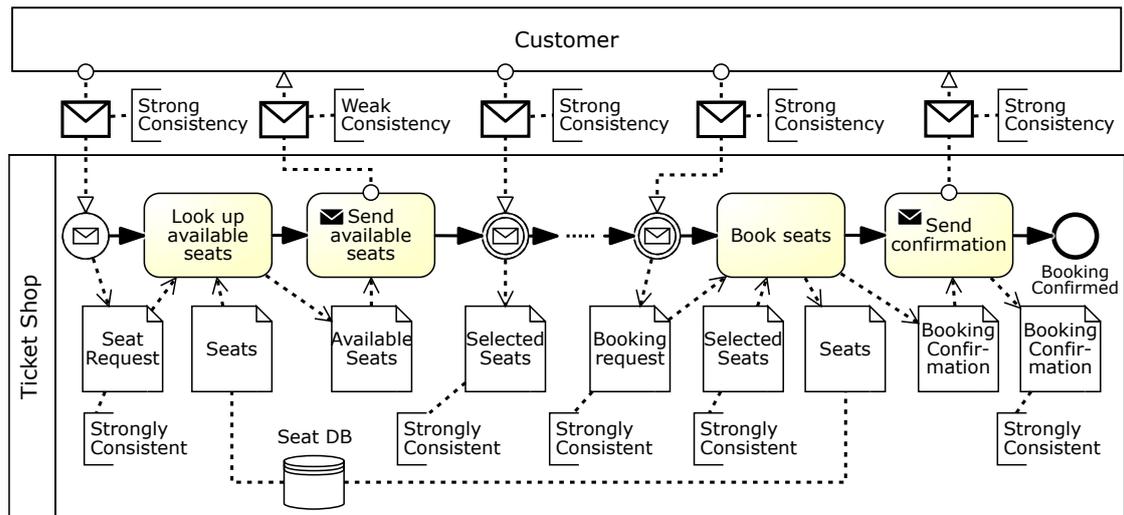
**Figure 4:** Data flow rules for senders and receivers based on consistency constraints represented by sample interactions from the ticket reservation choreography (details of the local control flow are omitted)

access restrictions with strongly consistent data objects, the annotation serves as an indicator for the process engineer to be aware of potential inconsistencies when designing the private process. Furthermore, inconsistencies arising from weak consistency may lead to conflicts. These conflicts need to be discovered in the local data flow of each participant individually. If the local data flow can lead to a conflict, additional interactions may need to be included in the choreography to introduce compensation behavior.

Interactions associated with event-bound strong consistency are initiated with the same restrictions as with the strong consistency constraint. However, the predefined events allow the sender to release the data objects from the restrictions, as shown by the ‘Price Offer’ data object in Figure 4. Moreover, a locked persistent data object can be unlocked and modified after the event occurred. Thus, in contrast to strong consistency, using write locks for persistent data objects exchanged with event-bound strong consistency can be beneficial as they allow safe access to the data objects after they are released. Accordingly, persistent data objects must not be modified in the period between the sending of the message and the event’s occurrence. On the receiving end, the data objects change from strongly consistent to weakly consistent after the occurrence of one of the specified events, as illustrated by the signal event in Figure 4.

## 4. Evaluation

To evaluate the proposed concept, its applicability to the motivating example proposed in Section 2.4 is assessed below. Furthermore, the framework’s benefits and limitations are discussed.



**Figure 5:** Excerpt of the private data flow of the ticket shop in the ticket reservation choreography

#### 4.1. Ticket Reservation Scenario

The evaluation focuses on the ticket reservation choreography illustrated in Figure 1. Considering that multiple customers book tickets concurrently, the ticket shop plans to only guarantee weak consistency when sharing the available seats with each customer to increase throughput. In the following, an excerpt of the ticket shop’s private process is enriched with the proposed extensions as depicted in Figure 5 to illustrate the interplay between private data flow and consistency constraint specifications.

By sharing the available seats with weak consistency, the ticket shop can access the persistent ‘Seats’ data object without a lock, which potentially increases the desired throughput for concurrent process instances. Since the ticket shop communicates the degree of consistency of the data on the public process, the customer is aware of potential inconsistencies and can take this into account for further decisions. In case the customer decides to book the ticket, the ticket shop needs to modify the ‘Seats’ data object with the ‘Book seats’ activity in order to create the booking confirmation. Since the data object ‘Seats’ is not locked, the write operation may fail, which stalls the execution. To handle this conflict, the corresponding activity requires an interrupting boundary event introducing compensation behavior, for example, by proposing alternative seats to the customer. Therefore, the decisions in data management of the ticket shop also require adapting the public process to ensure interoperability. However, by using the extended semantics as well as the compatibility rules introduced by this framework, potential conflicts can already be detected at design time.

#### 4.2. Discussion

By enriching choreography models with consistency constraints, process engineers gain a means of communication for interorganizational data management. Since the additional semantics are incorporated through annotations, the resulting model conforms to the BPMN standard.

Furthermore, data flow verification is kept local to each participant, so details about private process behavior do not need to be shared with other choreography participants. As a result, the framework allows for an iterative refinement of the choreography, as private processes need to be adapted to the consistency specifications and the adaptations may change the public process, which in turn may again have an impact on the private processes. During the refinement, only information about the public process needs to be shared among participants.

Despite that, developing a consistent data flow depends on an accurate representation of the choreography and data management. This requires that detailed information about data management is known at design time (e.g., the scope of a data object) and that the implemented data management must match the data management specified at design time. Currently, the framework is also limited to interaction behavior between two participants. The application of the framework to more complex interactions between multiple participants needs to be further explored, as, for example, data passed to a third party may inherit consistency constraints from the original sender and the intermediary.

Another aspect that needs to be considered is the cognitive load imposed by the extensions. To mitigate the impact, the number of annotations required could be reduced by establishing a default behavior for when no annotations are made. For example, in the choreography diagram, any interaction without annotation could be expected to follow the strong consistency constraint, as this provides the most reliable data management.

In addition, it should be noted that the framework focuses on the interplay of local data management and public consistency constraints and does not check for general data flow anomalies at the private process level. To this end, the framework could be combined with existing techniques that provide data flow verification [3, 4]. Nevertheless, by enabling iterative interorganizational alignment of each individual data management at the public process level, the framework allows for the design of more reliable process choreographies.

## 5. Related Work

To provide a brief overview of approaches related to the modeling framework presented, this section focuses on the existing works in the areas of data consistency and interorganizational data flow modeling. In literature, data inconsistencies are considered as a relevant data flow anomaly that can lead to conflicts in process execution. [4] identifies data inconsistencies as one of seven fundamental data flow validation issues that must be addressed during process design. To address this deficiency, [3] presents a data flow verification framework that enables the discovery of data flow anomalies by analyzing data dependencies within process execution. The framework also discusses conflicting data in terms of data consistency. However, the authors do not consider data access by concurrent process instances or distributed access across multiple participants.

Another approach to address data inconsistencies is the use of transactional behavior. In [5], Alonso et al. propose the use of workflow systems for the implementation of traditional transaction models. One of the main findings of their work is that workflow systems lack exception handling capabilities to enable transactional features. With our framework, we aim to enable the detection of potential exceptions due to data inconsistencies, which can then be

resolved at design time by introducing compensation behaviors. In addition, there are several approaches to integrate transactional behavior into business processes [10, 11]. However, the transactional properties may lead to rollbacks at runtime in case of exceptions, which can result in costs. By identifying potential exceptions at design time, progressive compensation behavior can be introduced that can avoid costly rollbacks during execution. The different combinations of process-related and transactional concepts are categorized by a taxonomy proposed in [12].

Although the literature already addresses the integration of data flow in the modeling of interorganizational processes, the consideration of data consistency in process design has received little attention in research. Meyer et al. [13] introduce a notion to automate the exchange of local data objects using a global data model. While the approach supports message correlation in multi-instance scenarios, concurrent access to exchanged data is not considered. Another approach proposed by Hahn et al. [14] decouples the data flow from the message flow. The interorganizational data flow is implicitly coordinated during execution by a middleware. However, their approach requires the participants to disclose parts of their private processes during design and the resolution of conflicts resulting from concurrent operations is not specified.

In [15], Haarmann et al. introduce a framework including formal semantics to describe and analyze data objects shared among process instances. While their framework involves concepts for data consistency preservation, it does not address data exchange between individual participants. Furthermore, the definition of constraints spanning multiple instances is described in [16]. Due to the lack of data-related information, applying the constraints to choreography diagrams may prove challenging. Finally, Kopp et al. [17] introduce choreography spheres to ensure transactional behavior across activities of different processes. Still, selecting an adequate scope of the spheres may require data-specific information in the model.

## 6. Conclusion

Inconsistent views of shared data can lead to conflicting interaction behavior during process choreography execution. Therefore, the preservation of data consistency must be considered during choreography design. As current choreography modeling languages provide limited data modeling capabilities, this paper proposes a modeling framework to enable the coordination of data consistency in process choreographies. To this end, the framework introduces the concept of consistency constraints for public processes to communicate the degree of data consistency that is guaranteed for each interaction. In addition, the compatibility of the private data flow with the consistency constraints on the public process can be examined locally using data accessibility specifications. As a result, choreography participants can coordinate their data management without revealing detailed information about their private processes. Currently, the framework only supports the coordination of two interacting participants. For future work, the framework can be extended to support choreographies with more participants. In addition, an automated verification of the private data flow's compatibility with the corresponding consistency constraints could facilitate the identification of conflicting behavior. Since the current extension relies on textual annotations, the use of a graphical notion may be evaluated for the extension. Eventually, more fine-grained consistency constraints can be added to increase the flexibility in data management design.

## References

- [1] M. Weske, *Business Process Management - Concepts, Languages, Architectures*, Third Edition, Springer, 2019. doi:10.1007/978-3-662-59432-2.
- [2] A. S. Tanenbaum, M. van Steen, *Distributed Systems - Principles and Paradigms*, 2nd Edition, Pearson Education, 2007.
- [3] S. X. Sun, J. L. Zhao, J. F. N. Jr., O. R. L. Sheng, Formulating the Data-Flow Perspective for Business Process Management, *Inf. Syst. Res.* 17 (2006) 374–391. doi:10.1287/isre.1060.0105.
- [4] S. W. Sadiq, M. E. Orłowska, W. Sadiq, C. Foulger, Data Flow and Validation in Workflow Modelling, in: K. Schewe, H. E. Williams (Eds.), *Database Technologies 2004, Proceedings of the Fifteenth Australasian Database Conference, ADC 2004, Dunedin, New Zealand, 18-22 January 2004*, volume 27 of *CRPIT*, Australian Computer Society, 2004, pp. 207–214. URL: <http://crpit.scem.westernsydney.edu.au/abstracts/CRPITV27Sadiq.html>.
- [5] G. Alonso, D. Agrawal, A. E. Abbadi, M. Kamath, R. Günthör, C. Mohan, Advanced Transaction Models in Workflow Contexts, in: S. Y. W. Su (Ed.), *Proceedings of the Twelfth International Conference on Data Engineering*, February 26 - March 1, 1996, New Orleans, Louisiana, USA, IEEE Computer Society, 1996, pp. 574–581. doi:10.1109/ICDE.1996.492208.
- [6] Object Management Group, *Business Process Model and Notation (BPMN)*, Version 2.0.2, 2014. URL: <https://www.omg.org/spec/BPMN/2.0.2>.
- [7] J. Gray, A. Reuter, *Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, 1993.
- [8] S. Gilbert, N. A. Lynch, Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services, *SIGACT News* 33 (2002) 51–59. doi:10.1145/564585.564601.
- [9] A. Meyer, L. Pufahl, D. Fahland, M. Weske, Modeling and Enacting Complex Data Dependencies in Business Processes, in: F. Daniel, J. Wang, B. Weber (Eds.), *Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings*, volume 8094 of *Lecture Notes in Computer Science*, Springer, 2013, pp. 171–186. doi:10.1007/978-3-642-40176-3\_14.
- [10] S. Dalal, S. Temel, M. C. Little, M. Potts, J. Webber, Coordinating Business Transactions on the Web, *IEEE Internet Comput.* 7 (2003) 30–39. doi:10.1109/MIC.2003.1167337.
- [11] M. P. Papazoglou, Web Services and Business Transactions, *World Wide Web* 6 (2003) 49–91. doi:10.1023/A:1022308532661.
- [12] P. W. P. J. Grefen, Transactional Workflows or Workflow Transactions?, in: A. Hameurlain, R. Cicchetti, R. Traunmüller (Eds.), *Database and Expert Systems Applications, 13th International Conference, DEXA 2002, Aix-en-Provence, France, September 2-6, 2002, Proceedings*, volume 2453 of *Lecture Notes in Computer Science*, Springer, 2002, pp. 60–69. doi:10.1007/3-540-46146-9\_7.
- [13] A. Meyer, L. Pufahl, K. Batoulis, D. Fahland, M. Weske, Automating data exchange in process choreographies, *Inf. Syst.* 53 (2015) 296–329. doi:10.1016/j.is.2015.03.008.
- [14] M. Hahn, U. Breitenbücher, F. Leymann, M. Wurster, V. Yussupov, Modeling Data Transformations in Data-Aware Service Choreographies, in: *22nd IEEE International Enterprise*

- Distributed Object Computing Conference, EDOC 2018, Stockholm, Sweden, October 16-19, 2018, IEEE Computer Society, 2018, pp. 28–34. doi:10.1109/EDOC.2018.00014.
- [15] S. Haarmann, M. Weske, Correlating Data Objects in Fragment-Based Case Management, in: W. Abramowicz, G. Klein (Eds.), Business Information Systems - 23rd International Conference, BIS 2020, Colorado Springs, CO, USA, June 8-10, 2020, Proceedings, volume 389 of *Lecture Notes in Business Information Processing*, Springer, 2020, pp. 197–209. doi:10.1007/978-3-030-53337-3\_15.
- [16] M. Leitner, J. Mangler, S. Rinderle-Ma, Definition and Enactment of Instance-Spanning Process Constraints, in: X. S. Wang, I. F. Cruz, A. Delis, G. Huang (Eds.), Web Information Systems Engineering - WISE 2012 - 13th International Conference, Paphos, Cyprus, November 28-30, 2012. Proceedings, volume 7651 of *Lecture Notes in Computer Science*, Springer, 2012, pp. 652–658. doi:10.1007/978-3-642-35063-4\_49.
- [17] O. Kopp, M. Wieland, F. Leymann, Towards choreography transactions, in: O. Kopp, N. Lohmann (Eds.), 1st Central-European Workshop on Services and their Composition, ZEUS 2009, Stuttgart, Germany, March 2-3, 2009. Proceedings, volume 438 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2009, pp. 49–54. URL: <http://ceur-ws.org/Vol-438/paper8.pdf>.