# Realizability is Controllability

Niels Lohmann and Karsten Wolf

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

**Abstract.** A *choreography* describes the interaction between services. It may be used for specification purposes, for instance serving as a contract in the design of an inter-organizational business process. Typically, not all describable interactions make sense which motivates the study of the *realizability* problem for a given choreography.
In this paper, we show that realizability can be traced back to the problem of *controllability* which asks whether a service has compatible partner processes. This way of thinking makes algorithms for controllability available for reasoning about realizability. In addition, it suggests alternative definitions for realizability. We discuss several proposals for defining realizability which differ in the degree of coverage of the specified interaction.

## 1 Introduction

When designing an inter-organizational business process, the involved parties (e.g., enterprises or business units) need to agree on many aspects of their interaction. One of these aspects is the order of exchanged messages between the parties. To this end, *choreographies* have been proposed. A choreography specification aims at specifying the interaction without revealing unnecessary details about the internal control flow of the involved parties. Keeping internals secret may have several reasons. On one hand, trade secrets may be involved as the parties may be competitors. On the other hand, an internal control flow may not exist when the choreography is specified in a design-by-contract scenario.

Several languages have been proposed for specifying choreographies (see [1] for a survey). They all have in common that they permit to specify unreasonable interactions. An example for a potentially unreasonable interaction is to require that a message from party $A$ to party $B$ must be exchanged before another message from $C$ to $D$. As long as no other messages are passed between $A$ and $C$ or $B$ and $C$ this requirement cannot be satisfied. For distinguishing between reasonable and unreasonable interaction, the concept of *realizability* was introduced for example in [2, 3].

In this paper, we address the following issues in existing approaches to choreographies and realizability notions. *First*, several approaches seem to focus on synchronous interaction. Asynchronous interaction is either not considered at all, or is brought into the approach as a derivative of the synchronous approach. For instance, some approaches specify only the order in which messages are sent but say nothing about the order in which they should be received. Consequently, we propose a formalism for modeling choreographies where synchronous and asynchronous communications are both first-class citizens. In our setting, causality between the receipt of a message and sending another one can be specified. *Second*, there appear to be several proposals for defining realizability. Consequently, we propose a hierarchy of realizability notions that includes and extends existing concepts. The hierarchy is systematically obtained by relating the problem of realizability to the problem of *controllability* in the sense of [4].

Controllability asks whether a given service has compatible partners. Existing techniques for answering the controllability problem are capable of synthesizing a compatible partner if it exists. Hence, *third*, we suggest techniques to synthesize internals of realizing partners. By relating the realizability problem to controllability, we, *fourth*, get the opportunity to study specifications that involve both a choreography and the specification of the internal behavior of some of the parties. This way, we marry the choreography approach with the orchestration approach. Both approaches have so far been conceived as complementary paradigms for building up complex processes from services.

The rest of this paper is organized as follows. In Sect. 2, we introduce a formal framework which allows us to reason about choreographies in a formal and language-independent manner. In Sect. 3, we recall different realizability notions and introduce the novel concept of *distributed realizability*, which seamlessly complements existing notions. The main contribution of the paper is presented in Sect. 4: the realizability problem can be formulated in terms of controllability and algorithms for controllability can be used to prove realizability by synthesizing realizing services. Section 5 is dedicated to issues arising when asynchronous communication is considered. In Sect. 6, we show how the relationship between controllability and realizability can be used to combine aspects from interaction modeling and interconnected models. Section 7 discusses related work and Sect. 8 concludes the paper and gives directions for future research.

## 2 A Formal Framework for Choreographies

To formally reason about choreographies, we first introduce a formal framework that employs automata to model single services as well as whole service choreographies. Throughout this paper, fix a finite set of message channels $M$ that is partitioned into asynchronous message channels $M_A$ and synchronous message channels $M_S$. From $M$, derive a set of message events $E := {!E} \cup {?E} \cup {!?E}$, consisting of asynchronous send events $!E := \{!x \mid x \in M_A\}$, asynchronous receive events $?E := \{?x \mid x \in M_A\}$, and synchronization events $!?E := \{!?x \mid x \in M_S\}$. Furthermore, we distinguish a non-communicating event $\tau \notin E$. In this paper, we assume that asynchronous messages may overtake each other. We claim that this is — compared to FIFO queues for communicating finite state machines [5] — a more natural approach to model asynchronicity, because it makes less assumptions about the underlying infrastructure.

**Definition 1 (Peer, collaboration).** *A* peer $P = [I, O]$ *consists of a set of input message channels* $I \subseteq M$ *and a set of output message channels* $O \subseteq M$, $I \cap O = \emptyset$. *A* collaboration *is a set* $\{[I_1, O_1], \ldots, [I_n, O_n]\}$ *of peers such that* $I_i \cap I_j = \emptyset$ *and* $O_i \cap O_j = \emptyset$ *for all* $i \neq j$, *and* $\bigcup_{i=1}^n I_i = \bigcup_{i=1}^n O_i$.

A collaboration is a set of bilaterally communicating peers and can be seen as the structure or syntactic signature of a choreography, because the internal behavior of the peers is left unspecified and only their message channels are given. Figure 1(a) shows the graphical notation we use in this paper to depict collaborations. The desired observable behavior of a collaboration can be specified with a finite state automaton whose transitions are labeled with message events.

**Definition 2 (Peer automaton).** *A* peer automaton $A = [Q, \delta, q_0, F, \mathcal{P}]$ *is a tuple such that* $Q$ *is a finite set of states,* $\delta \subseteq Q \times (E_I \cup E_O \cup \{\tau\}) \times Q$ *is a transition relation,* $q_0 \in Q$ *is an initial state,* $F \subseteq Q$ *is a set of final states, and* $\mathcal{P} = \{[I_1, O_1], \ldots, [I_n, O_n]\}$ *is a nonempty set of peers. Thereby,* $E_I := \{?x \mid x \in M_A \cap \bigcup_{i=1}^n I_i\} \cup \{!?x \mid x \in M_S \cap \bigcup_{i=1}^n I_i\}$ *are the input events of* $A$ *and* $E_O := \{!x \mid x \in M_A \cap \bigcup_{i=1}^n O_i\} \cup \{!?x \mid x \in M_S \cap \bigcup_{i=1}^n O_i\}$ *are output events of* $A$.

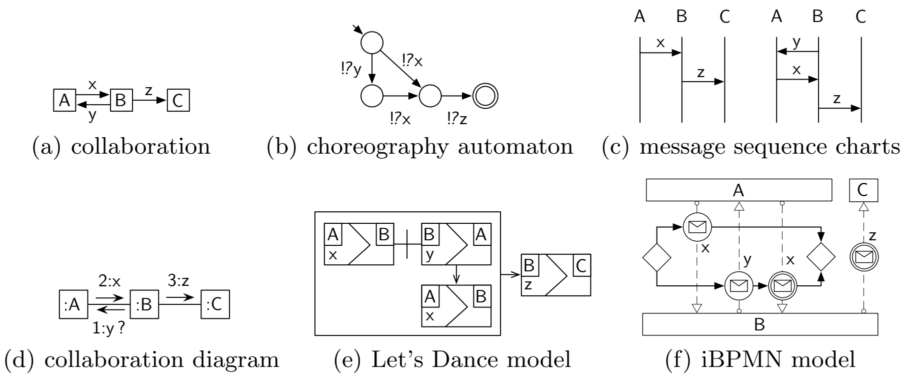
*A* implements *the peers* $\mathcal{P}$, *and for* $(q, x, q') \in \delta$, *we also write* $q \xrightarrow{x} q'$. *A is called a* single-peer automaton, *if* $|\mathcal{P}| = 1$. *A is called a* multi-peer automaton, *if* $|\mathcal{P}| > 1$ *and* $\mathcal{P}$ *is a collaboration. A is called* $\tau$-free *if* $q \xrightarrow{x} q'$ *implies* $x \neq \tau$ *for all* $q, q' \in Q$. *A* run *of A is a sequence of events* $x_1 \cdots x_m$ *such that* $q_0 \xrightarrow{x_1} \cdots \xrightarrow{x_m} q_f$ *with* $q_f \in F$.

We use the standard graphical representation for automata (cf. Fig. 1(b)). A $\tau$-free peer automaton can be used to define a choreography.

**Definition 3 (Choreography).** *Let* $A = [Q, \delta, q_0, F, \mathcal{P}]$ *be a multi-peer automaton. A run* $\rho$ *of A is a* conversation *if no* $\tau$ *transition occurs in* $\rho$ *and, for all* $x \in M_A$, $\#_{!x}(\rho) = \#_{?x}(\rho)$ *and for every prefix* $\rho'$ *of* $\rho$ *holds:* $\#_{!x}(\rho') \geq \#_{?x}(\rho')$. *Thereby,* $\#_x(\rho)$ *denotes the number of occurrences of the message event* $x$ *in the run* $\rho$. *A* choreography *is a set of conversations. For a run* $\rho$, *define the event sequence of* $\rho$ *as* $\rho_{|E}$ *(i.e.,* $\rho$ *without* $\tau$*-steps). The* language *of A, denoted* $\mathcal{L}(A)$, *is the union of the event sequences of all runs of A. A is a* choreography automaton, *if A is* $\tau$*-free and* $\mathcal{L}(A)$ *is a choreography.*

The requirements for a conversation state that asynchronous events are always paired (messages do not get lost), and a send event always occurs before the respective receive event.

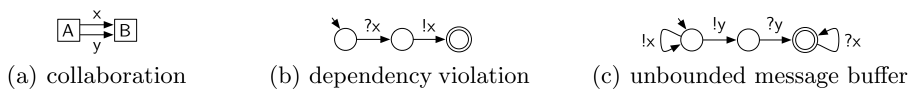
A mapping from existing interaction modeling languages such as interaction Petri nets [6], Let's Dance [7], message sequence charts [3], collaboration diagrams [8], or iBPMN [9] to choreography automata is straightforward. While these languages differ in syntax and semantics, concepts such as an underlying collaboration (i.e., the peers and their message channels) and the choreography (i.e., the intended global behavior) can be easily derived from these languages. Figure 1 shows different models specifying the same globally observable behavior.



(a) collaboration (b) choreography automaton (c) message sequence charts

(d) collaboration diagram (e) Let's Dance model (f) iBPMN model

**Fig. 1.** Different models specifying the choreography $\{!?\mathsf{x}\,!?\mathsf{z}, !?\mathsf{y}\,!?\mathsf{x}\,!?\mathsf{z}\}$.



(a) collaboration (b) dependency violation (c) unbounded message buffer

**Fig. 2.** The multi-peer automata (b) and (c) do not specify a choreography.

We decided to use an automaton model, because it lacks structural restrictions and is naturally linked to regular languages.

However, not every $\tau$-free multi-peer automaton specifies a choreography. Figure 2(b) depicts a peer automaton that violates the causal dependency between an asynchronous send and the respective receive event. Another problem arises in settings such as shown in Fig. 2(c) in which an arbitrary number of x messages needs to be buffered. In Sect. 5, we will show how bounded message buffers can be enforced and all runs that are not conversations can be removed from a peer automaton.

Finally, not every choreography can be expressed by a multi-peer automaton, for example the context-free choreography $(!?\mathsf{d})^i(!?\mathsf{e})^i = \{\epsilon, !?\mathsf{d}!?\mathsf{e}, !?\mathsf{d}!?\mathsf{d}!?\mathsf{e}!?\mathsf{e}, \dots\}$. In this paper, we only consider regular choreographies, because language equivalence and language containment is undecidable for context-free languages, and hence realizability is undecidable for context-free choreographies.

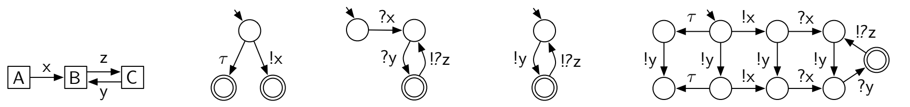
## 3 Realizability Notions

As discussed in the introduction, not every choreography can be implemented by peers. To relate the specified interactions of a choreography and the interactions between peers, we first define the behavior of composed single-peer automata.

In the composition, pending asynchronous messages are represented by a multiset. Denote the set of all multisets over $M_A$ with $\mathit{Bags}(M_A)$, the empty multiset with $[\,]$, and the multiset containing only one instance of $x \in M_A$ with $[x]$. Addition of multisets is defined pointwise.

**Definition 4 (Composition of single-peer automata).** *Let $A_1, \ldots, A_n$ be single-peer automata ($A_i = [Q_i, \delta_i, q_{0_i}, F_i, \{P_i\}]$ for $i = 1, \ldots, n$) such that their peers form a collaboration. Define the* composition $A_1 \oplus \cdots \oplus A_n$ *as the multi-peer automaton $[Q, \delta, q_0, F, \{P_1, \ldots, P_n\}]$ with $Q := Q_1 \times \cdots \times Q_n \times Bags(M_A)$, $q_0 := [q_{0_1}, \ldots, q_{0_n}, [\,]]$, $F := F_1 \times \cdots \times F_n \times \{[\,]\}$, and, for all $i \neq j$ and $B \in Bags(M_A)$ the transition relation $\delta$ contains exactly the following elements:*

- $[q_1, \ldots, q_i, \ldots, q_n, B] \xrightarrow{\tau} [q_1, \ldots, q_i', \ldots, q_n, B]$,
  *iff $[q_i, \tau, q_i'] \in \delta_i$ (internal move by $A_i$),*
- $[q_1, \ldots, q_i, \ldots, q_n, B] \xrightarrow{!x} [q_1, \ldots, q_i', \ldots, q_n, B + [x]]$,
  *iff $[q_i, !x, q_i'] \in \delta_i$ (asynchronous send by $A_i$),*
- $[q_1, \ldots, q_i, \ldots, q_n, B + [x]] \xrightarrow{?x} [q_1, \ldots, q_i', \ldots, q_n, B]$,
  *iff $[q_i, ?x, q_i'] \in \delta_i$ (asynchronous receive by $A_i$), and*
- $[q_1, \ldots, q_i, \ldots, q_j, \ldots, q_n, B] \xrightarrow{!?x} [q_1, \ldots, q_i', \ldots, q_j', \ldots, q_n, B]$,
  *iff $[q_i, !?x, q_i'] \in \delta_i$ and $[q_j, !?x, q_j'] \in \delta_j$ (synchronization between $A_i$ and $A_j$).*



(a) collaboration (b) peer A (c) peer B (d) peer C (e) composition A ⊕ B ⊕ C

**Fig. 3.** Composition of single peer automata (unreachable states are omitted).

The composition of single-peer automata yields a multi-peer automaton (see Fig. 3 for an example). Its behavior can be related to a specified choreography which leads to the concept of complete realizability [1–3, 6, 8].

**Definition 5 (Complete realizability).** *Let $C$ be a choreography automaton implementing the peers $\{P_1, \ldots, P_n\}$. The single-peer automata $A_1, \ldots, A_n$* completely realize *$C$ if, for all $i$, $A_i$ implements $\{P_i\}$ and $\mathcal{L}(A_1 \oplus \cdots \oplus A_n) = \mathcal{L}(C)$.*

Complete realizability is a strong requirement, because it demands that the observable behavior of the endpoints exactly matches the choreography. In reality, it is often the case that not all aspects of a choreography can be implemented. To this end, Zaha et al. [10] introduce the notion *local enforceability* (also called *partial realizability* or *weak realizability*) which only demands that a subset of the choreography is realized by the peer implementations:

**Definition 6 (Partial realizability).** *Let $C$ be a choreography automaton implementing the peers $\{P_1, \ldots, P_n\}$. The single-peer automata $A_1, \ldots, A_n$* partially realize *$C$ if, for all $i$, $A_i$ implements $\{P_i\}$ and $\emptyset \neq \mathcal{L}(A_1 \oplus \cdots \oplus A_n) \subseteq \mathcal{L}(C)$.*

Obviously, complete realizability implies partial realizability. Though this weaker notion ensures that all constraints of the choreography are fulfilled, it

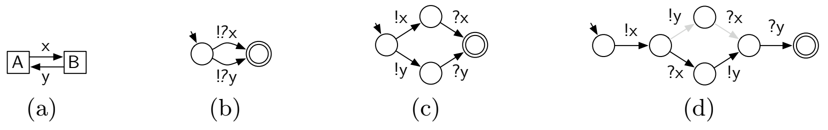


**Fig. 4.** For the collaboration (a), the choreography (b) is completely realizable, (c) is distributedly realizable, and (d) is partially realizable.

still only considers a single tuple of peer automata. If there does not exist such tuple of automata that realizes the *complete* choreography, there might still exist a *set* of tuples — each partially realizing the choreography — which distributedly realizes the complete choreography:

**Definition 7 (Distributed realizability).** *Let $C$ be a choreography automaton implementing the peers $\{P_1, \ldots, P_n\}$. The tuples of single-peer automata $[A_{1_1}, \ldots, A_{n_1}], \ldots,$ $[A_{1_m}, \ldots, A_{n_m}]$* distributedly realize *$C$ if, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, (i) $A_{i_j}$ implements $\{P_i\}$, (ii) $\emptyset \neq \mathcal{L}(A_{1_j} \oplus \cdots \oplus A_{n_j}) \subseteq \mathcal{L}(C)$, and (iii) $\bigcup_{j=1}^{m} \mathcal{L}(A_{1_j} \oplus \cdots \oplus A_{n_j}) = \mathcal{L}(C)$.*

Distributed realizability allows for design time coordination between peers: From a set of different possible implementations, we can choose a specific tuple of peer implementations that are coordinated in the sense that each peer can rely on the other peer's behavior. In addition, every conversation that is specified by the choreography can be realized by at least one tuple of implementing peers; that is, the choreography does not contain "dead code". While being a stronger notion than partial realizability (i.e., more of the choreography's behavior is implemented), it is still a weaker notion than complete realizability.

As an example, consider the collaboration depicted in Fig. 4(a). The choreography in which the peers communicate synchronously (b) is completely realizable by a set of peers which synchronize at runtime via message x or y. In case the messages are sent asynchronously (c), this is no longer possible. This choreography is not completely realizable, because there does not exist a single pair of peer automata that implement the specified behavior. However, the implementations can be coordinated at design time: either peer A sends a message and peer B is quiet or the other way around. These two pairs distributedly realize the whole choreography. Finally, choreography (d) can only be partially realized, because the conversation !x!y?x?y cannot be implemented by the peers without also producing the unspecified conversations !y!x?x?y or !y!x?y?x.

## 4 From Choreographies to Orchestrations

In this section, we link realizability to controllability [4], a correctness criterion that was originally defined for service orchestrations [11]. We first recall how choreography conformance can be checked using a monitor. Then, we derive an orchestration service from this monitor and show how its partner services are related to peer implementations. Finally, we show how the algorithm to check controllability can be used to synthesize peer implementations.

## 4.1 Choreography Monitor Service

Choreography realization plays an important rule in inter-organizational business processes where a choreography is used to specify a business protocol the parties agreed to follow. In this setting, it is not sufficient to prove realizability of the choreography alone, but also to constantly monitor whether the agreed protocol is followed by the peers. In that setting, an *enterprise service bus* [12] is used to provide the connection between the individual peers. It monitors the message exchanges between the peers. Protocol violations can then be later prosecuted by examining log files or even during runtime as proposed in [13].

In this setting, it is important that the interactions are monitored in an unobtrusive way; that is, the interactions must not be altered by the monitor and the peers must not be aware of the monitor. By definition, the choreography automaton exactly determines the desired interactions. However, it blocks unspecified interactions and hence lacks the monitor property. To this end, the states of the choreography automaton needs to be made *deterministic*. This is a standard operation for regular automata and does not restrict generality. It ensures that in every state $q$ and for each event $x$ there is exactly one $x$-labeled edge leaving $q$. In case such a transition was not specified, the new introduced edge leads to a non-final deadlock state. A composition of peers is monitored as follows.

**Definition 8 (Monitored composition).** *Let $C = [Q_C, \delta_C, q_{0_C}, F_C, \mathcal{P}]$ be a deterministic choreography automaton with $\mathcal{P} = \{P_1, \ldots, P_n\}$ and let $A_1, \ldots, A_n$ be single-peer automata ($A_i = [Q_i, \delta_i, q_{0_i}, F_i, \{P_i\}]$ for $i = 1, \ldots, n$) such that their peers form a collaboration. Define the* monitored composition $C \otimes (A_1 \oplus \cdots \oplus A_n)$ *as the multi-port peer automaton $[Q, \delta, q_0, F, \mathcal{P}]$ with $Q := Q_C \times Q_1 \times \cdots \times Q_n \times Bags(M_A)$, $q_0 := [q_{0_C}, q_{0_1}, \ldots, q_{0_n}, []]$, $F := F_M \times F_1 \times \cdots \times F_n \times []]$, and, for all $i \neq j$ and $Bags(M_A)$, the transition relation $\delta$ contains exactly the following elements:*

- $[q, q_1, \ldots, q_i, \ldots, q_n, B] \xrightarrow{\tau} [q, q_1, \ldots, q_i', \ldots, q_n, B]$, *iff* $[q_i, \tau, q_i'] \in \delta_i$ *(internal move by $A_i$, invisible to $C$),*
- $[q, q_1, \ldots, q_i, \ldots, q_n, B] \xrightarrow{!x} [q', q_1, \ldots, q_i', \ldots, q_n, B + [x]]$, *iff* $[q_i, !x, q_i'] \in \delta_i$ *and* $[q, !x, q'] \in \delta_C$ *(asynchronous send by $A_i$, monitored by $C$),*
- $[q, q_1, \ldots, q_i, \ldots, q_n, B + [x]] \xrightarrow{?x} [q', q_1, \ldots, q_i', \ldots, q_n, B]$, *iff* $[q_i, ?x, q_i'] \in \delta_i$ *and* $[q, !?x, q'] \in \delta_C$ *(asynchronous receive by $A_i$, monitored by $C$),*
- $[q, q_1, \ldots, q_i, \ldots, q_j, \ldots, q_n, B] \xrightarrow{!?x} [q', q_1, \ldots, q_i', \ldots, q_j', \ldots, q_n, B]$, *iff* $[q_i, !?x, q_i'] \in \delta_i$, $[q_j, !?x, q_j'] \in \delta_j$, *and* $[q, !?x, q'] \in \delta_C$ *(synchronization between $A_i$ and $A_j$, monitored by $C$).*

The monitor synchronizes with the message events of the single-peer automata, but does not constrain their behavior. The monitor only has an effect on the final states of the composition. Only if all single-peer automata *and* the monitor reach a final state, this state is final in the monitored composition.

We can now change the point of view and regard the monitor as a service that is communicating with several other services by synchronous message events. Again, this service will reach a final state iff the message events from the environment are observed in the correct order. Note that a choreography only considers message *events* (i.e., sending or receipt of a message) rather than the messages itself (e.g., asynchronously sent messages that are pending on a channel). Hence, all message events of the monitor service automaton are synchronous.

**Definition 9 (Monitor service).** *Let $C = [Q, \delta, q_0, F, \mathcal{P}]$ be a deterministic choreography automaton. Define the* monitor service $M_C := [Q, \delta_M, q_0, F, \mathcal{P}]$ *with the transition relation $\delta_M \subseteq Q \times \{!?\langle x\rangle \mid x \in E\} \times Q$ with $[q, !?\langle x\rangle, q'] \in \delta_M$ iff $(q, x, q') \in \delta$.*

The monitor service of a choreography can now be interpreted as an orchestrator that communicates synchronously with its peers. The introduction of a synchronous event $!?\langle x\rangle$ for event $x$ is a technical necessity to "encode" the original nature of the event $x$. In the final peer implementations, we will later replace the message event $!?\langle x\rangle$ by $x$ again. For example, an asynchronous event $?a$ is monitored as $!?\langle ?a\rangle$.

## 4.2 Link to Controllability

In this section, we show how the different realizability notions are related to controllability [4]. Controllability is a correctness criterion for services: a service $A$ is controllable iff there exists a compatible service $B$ (i.e., $A \oplus B$ is deadlock free). Controllability can be extended to multi-port services.

**Definition 10 (Decentralized controllability [4]).** *Let $A$ be a multi-peer automaton implementing the peer $\{[I_1, O_1], \ldots, [I_n, O_n]\}$. A is* decentralized controllable *iff there exists a tuple of single-peer automata $[B_1, \ldots, B_n]$ such that $B_i$ implements the peer $\{[O_i, I_i]\}$ and $A \oplus B_1 \oplus \cdots \oplus B_n$ is deadlock free; that is, every reachable state $q \notin F$ has a successor.*

We call $[B_1, \ldots, B_n]$ a *strategy* of $A$. Note that the single-peer automata $B_1, \ldots, B_n$ only communicate with $A$ and do not share message channels. Hence, they cannot communicate directly with each other during runtime. Only during design time of $B_1, \ldots, B_n$ it is possible to coordinate their behavior.

While realizability notions require the existence of single-peer automata whose composition realizes a certain parts of a given choreography, decentralized controllability requires the existence of single-peer automata which, when composed to a given service, result in deadlock-free communication. When considering the monitor automaton of a choreography, controllability and realizability coincide which is the main result of this paper:

**Theorem 1 (Realizability is controllability).** *Let $C$ be a choreography automaton implementing the peers $\{P_1, \ldots, P_n\}$ and $M_C$ a monitor service automaton for $C$.*

*(1) $C$ is partially realizable iff $M_C$ is decentralized controllable.*
*(2) $C$ is distributedly realizable iff $M_C$ is decentralized controllable and for the set of strategies $\mathcal{S}$ holds: $\bigcup_{[A_1,\ldots,A_n]\in\mathcal{S}} \mathcal{L}(A_1 \oplus \cdots \oplus A_n) = \mathcal{L}(C)$.*
*(3) $C$ is completely realizable iff $M_C$ is decentralized controllable and there exists a strategy $[A_1, \ldots, A_n]$ such that $\mathcal{L}(A_1 \oplus \cdots \oplus A_n) = \mathcal{L}(C)$.*

The proof of Thm. 1 follows immediately from the definition of decentralized controllability, the definition of the monitor service (any unspecified behavior will lead to a deadlock) and the definitions of partial, distributed, and complete realizability.

Theorem 1 links several notions of realizability, the central correctness criterion for choreographies, to controllability. The latter was originally proposed as a "soundness notion for services" and was used to analyze service orchestrations [11].

### 4.3 Synthesizing Realizing Peers

In the remainder of this section, we sketch the algorithm from [4] to check for decentralized controllability. It consists of four steps: (1) peer overapproximation, (2) removal of reachable deadlocks, (3) resolution of dependencies between peers, and (4) peer projection. We will explain the steps in more detail below.

Firstly, an over-approximation of the possible interactions with the given multi-peer automaton is calculated. This step is necessary in the setting of asynchronous communication, because the decoupling of sending and receiving actions limits the observability of actions. When considering a monitor automaton, we can skip this step, because the monitor automaton communicates entirely synchronously (cf. Def. 9).

In a second step, all reachable deadlocks and states from which no final state is reachable are removed. Thereby, a deadlock is considered unreachable if the event leading to it is impossible in the respective state. An example would be a receipt of an asynchronous message in the initial state, for instance an edge labeled with "!?⟨?x⟩". Keeping these states is a technical necessity and is further discussed in [4].

From the remaining automaton, we have to make sure that message events that cannot be coordinated by the peers are independent, meaning they can occur in any order. We call two message events *distant* if there exists no peer that can observe both:

**Definition 11 (Distant message events).** *Let $A = [Q, \delta, q_0, F, \mathcal{P}]$ be a choreography automaton. Two message events $a, b \in E$ are* distant *iff there exist no peer $[I, O] \in \mathcal{P}$ such that $\{a, b\} \subseteq (I \cup O)$.*

No we can define independence between distant events as follows.

**Definition 12 (Independence [4]).** *Let $A = [Q, \delta, q_0, F, \mathcal{P}]$ be a $\tau$-free multi-peer automaton and $a, b \in E$ be distant message events.*

- $a$ activates $b$ *in $q \in Q$, if there exist states $q_a, q_{ab} \in Q$ with $q \xrightarrow{a} q_a \xrightarrow{b} q_{ab}$, but there exists no state $q_b \in Q$ with $q \xrightarrow{b} q_b$.*
- $a$ disables $b$ *in $q \in Q$, if there exist states $q_a, q_b \in Q$ with $q \xrightarrow{a} q_a$, $q \xrightarrow{b} q_b$, but there exists no state $q_{ab} \in Q$ with $q_a \xrightarrow{b} q_{ab}$.*
- *Two states $q_1, q_2 \in Q$ are* equivalent *iff $\mathcal{L}([Q, \delta, q_1, F, \mathcal{P}]) = \mathcal{L}([Q, \delta, q_2, F, \mathcal{P}])$.*
- *$a$ and $b$ are* independent *iff, for all states $q \in Q$ holds: $a$ neither activates nor disables $b$ in $q$ and, if $q \xrightarrow{a} q_a \xrightarrow{b} q_{ab}$ and $q \xrightarrow{b} q_b \xrightarrow{a} q_{ba}$, then $q_{ab}$ and $q_{ba}$ are equivalent.*

These independence requirements are weaker than the *lossless-join* property proposed in [2] and the *well-informed* property proposed in [8] which both aim at complete realizability only. They are, however, very similar the *autonomous property* [2]. If all distant events are independent and the removal of deadlocking states did not yield an empty automaton, we can finally project the remaining multi-peer automaton to the single-peer automata.

**Definition 13 (Peer projection).** *Let $A = [Q, \delta, q_0, F, \mathcal{P}]$ be a multi-peer automaton and $[I, O] \in \mathcal{P}$ be a peer implemented by $A$. Define the projection of $A$ to the peer $[I, O]$, denoted $A_{|[I,O]}$, as the single-peer automaton $[Q', \delta', q_0', F', \{[I, O]\}]$ with the initial state $q_0' := closure_{[I,O]}(\{q_0\})$ and $Q'$, $\delta'$, $F'$ inductively defined as follows:*

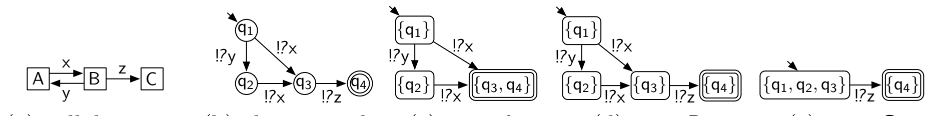
- *$q_0' \in Q'$.*
- *If $q \in Q'$ with $q_1 \in q$, $(q_1, x, q_2) \in \delta$, and $x \in I \cup O$, then $q' \in Q'$ with $q' := closure_{[I,O]}(q_2) \in Q'$ and $(q, x, q') \in \delta'$. $q' \in F'$ iff $q' \cap F \neq \emptyset$.*

*Thereby, define $closure_{[I,O]}(S) := \{q' \mid q \in S, q \xrightarrow{x_1 \cdots x_n} q', x_i \notin (I \cup O)\}$ for a set of states $S \subseteq Q$.*

The set $closure_{[I,O]}(S)$ contains all states reachable with a (possibly empty) sequence from a state of $S$ that does not contain an event from $(I \cup O)$. The definition is basically taken from the controllability decision algorithm [4] and was first proposed to be used as a projection algorithm by Decker in [14].

In a final step, we have to "restore" the original message model of the peer implementations that was set to synchronous communication in Def. 9. To this end, we replace each message event "$!?\langle x \rangle$" by "$x$" (e.g., the event $!?\langle ?a \rangle$ observed by the monitor is changed to $?a$ in the peer projection). This step from synchronous to possibly asynchronous is valid, because we ensured that any distant events are independent. Figure 5 shows an example.

In order to synthesize single-peer automata, potential dependencies between distant message events need to be resolved. The resolution of dependencies is the most important part of the synthesis algorithm for decentralized controllability. Independency can be achieved by removing those edges and states from the automaton that are dependent. In the case of disabling of events, this removal contains nondeterminism: if, for instance, an event $a$ disables an event $b$ in a state



(a) collaboration (b) choreography (c) peer A (d) peer B (e) peer C

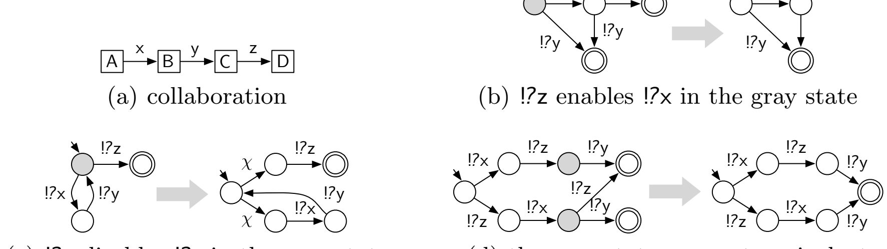
**Fig. 5.** A choreography and its projection to its peers.

$q$, we can decided whether to remove the $a$-successor or the $b$-successor of $q$ [4]. This mutually exclusive deletion yields two different tuples of implementing peers. In the following definition, we introduce a global decision event $\chi$ to express the different outcomes of this nondeterminism.

**Definition 14 (Resolution of dependency).** *Let $A = [Q, \delta, q_0, F, \mathcal{P}]$ be a $\tau$-free multi-peer automaton and $a, b \in E$ be distant message events.*

1. *If $a$ disables $b$ in a state $q \in Q$, then introduce two new states $q_a$ and $q_b$ with $q \xrightarrow{\chi} q_a$, $q \xrightarrow{\chi} q_b$ such that $q_a$ has all outgoing edges of $q$ that are not labeled with $b$ and $q_b$ has all outgoing edges of $q$ that are not labeled with $a$. Then remove all outgoing edges of $q$ that are not labeled with $\chi$.*
2. *If $a$ enables $b$ in a state $q \in Q$, then delete the state $q_{ab}$ with $q \xrightarrow{a} q_a \xrightarrow{b} q_{ab}$.*
3. *If the states $q_{ab}, q_{ba} \in Q$ with $q \xrightarrow{a} q_a \xrightarrow{b} q_{ab}$ and $q \xrightarrow{b} q_b \xrightarrow{a} q_{ba}$ are not equivalent, then delete $q_{ab}, q_{ba}$ and unite $A$ with the $\tau$-free multi-peer automaton $A' = [Q', \delta', q'_0, F', \mathcal{P}]$ with $\mathcal{L}(A') = \mathcal{L}([Q, \delta, q_{ab}, F, \mathcal{P}]) \cap \mathcal{L}([Q, \delta, q_{ba}, F, \mathcal{P}])$ and add the edges $q_a \xrightarrow{b} q'_0$ and $q_b \xrightarrow{a} q'_0$.*

The first step introduces the global decision events in case of disabling of an event. The second step removes states to avoid the enabling of an event. In the third step, equivalence of states that are reached by different interleavings of events is enforced by intersecting the runs reachable from these states. As we consider regular languages, the automaton having this intersection as language can be constructed easily.



(a) collaboration (b) !?z enables !?x in the gray state (c) !?x disables !?z in the gray state (d) the gray states are not equivalent

**Fig. 6.** Examples for the resolution of dependencies.

Figures 6(b)–(d) depict examples for each step. To increase legibility, the edges of the monitor services are labeled with the original event "$x$" instead of the encoded event "$!?\langle x \rangle$". Note that the removal of states and edges can introduce new deadlocks and make other states unreachable from the initial state. Such states need to be removed before projection. A multi-peer automaton is not decentralized controllable if all states are removed.

A multi-peer automaton with global decision events (cf. Fig. 6(c)) implicitly characterizes a set of multi-peer automata in which these decisions have been resolved. Each resolution of these decisions results in a tuple of implementing peers which can be derived using the projection defined in Def. 13. For the example of Fig. 6(c), the global decision is resolved independently each time the initial state is reached. Figure 7 depicts the different resolutions of the global decisions (again, "$x$" is used as label rather than "$!?\langle x \rangle$"). Each resolution represents a design-time coordination between the peers A and C on how often the !?x!?y loop should be traversed. As the peers A and C cannot communicate with each other, this coordination cannot be done during runtime. The set of all possible implementations distributedly realize the choreography.

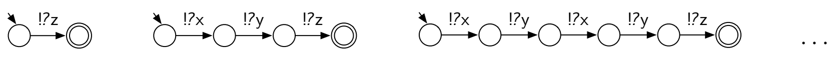


**Fig. 7.** Resolutions of the global decisions in the automaton depicted in Fig. 6(b).

With the presented approach, we are able to synthesize single-peer automata that control a given monitor service in a decentralized manner. Using Thm. 1, we can use the same algorithm to synthesize peers for the different realizability notions. It is worthwhile to mention that the approach aims at finding the strongest applicable realizability notion. In case a state need to be deleted due to dependencies (cf. Def. 14), we can derive diagnosis information:

– If a state is deleted by step 2 or 3, the choreography is neither completely realizable nor distributedly realizable.
– If a global decision is introduced by step 1, the choreography is not completely realizable, because the considered events are mutually exclusive.
– If the initial state is removed, the choreography is not partially realizable.

In any case, the respective state and the events that require state deletion can be used to diagnose the choreography and to introduce messages that restore independency.

## 5 Asynchronous Communication

Many interaction modeling languages (e.g., WS-CDL or interaction Petri nets) assume atomic and hence synchronous message exchange; that is, the sending and receiving of a message is specified to occur at the same time. Peers realizing such a choreography model inherit this synchronous message model. In implementations, however, asynchronous communication is often preferred over synchronous communication as a "fire and forget" send action is more efficient than a blocking handshaking.

To this end, we studied in [15] how synchronous peers that realize a choreography can be "desynchronized"; that is, atomic message exchange is decoupled to a pair of asynchronous send and receive actions. This desynchronization in turn might introduce deadlocks, and the correction towards deadlock freedom results in refinements of the choreography which require domain information and can hardly be automatized. Fu et al. [16] propose a reverse approach and study "synchronizability" of choreographies — a property under which asynchronous communication can be safely abstracted to synchronous communication. Synchronizability can help to detect problems introduced by asynchronous communication, but is only a sufficient criterion and offers only limited support in resolving these issues.

To avoid both restrictions during the design time of a choreography and a later change of the communication model, we allow to individually define, for each message, whether it should be transfered in an asynchronous or synchronous manner (cf. Def. 2). We claim that the nature of the message transfer is usually known in an early design phase and helps to refine the choreography model.

Unlike related work on collaboration diagrams or conversation protocols, we thereby do not just specify the order in which *send* events occur, but also describe the moment of the respective *receive* events. This is crucial to be able to specify dependencies between asynchronous messages. For instance, one is able to express that a customer must not send an order message to a shop before he received the terms of payment. If modeled synchronously, the shop would be blocked as long as the customer reads the terms of payment.

In addition, the precise specification of message receipt ensures that the message exchange between the peers can be realized with bounded message buffers. This is not only motivated by implementation issues, but also in the fact that unbounded queues would result in an infinite state automaton for which controllability and hence realizability would be undecidable [17].

In Def. 3, we restricted choreographies to only consist of conversations. This does not constrain synchronous message events, but only the asynchronous message events. The following definition manipulates an arbitrary multi-peer automaton such that every run is a conversation; that is, its collaboration language is a choreography. Furthermore, no run will exceed a given message bound $k$.

**Definition 15 ($k$-bounded peer automaton).** *Let $A = [Q, \delta, q_0, F, \mathcal{P}]$ be a peer automaton and $k \in \mathbb{N}$. Define the* $k$-bounded peer automaton *$A_k := [Q', \delta', q_0', F', \mathcal{P}]$ with $Q' := Q \times Bags_k(M_A)$, $q_0' := [q_0, []]$, $F' := F \times \{[]\}$ and $\delta'$ contains exactly the following elements ($B \in Bags_k$):*

- $\big[[q, B], !?x, [q', B]\big] \in \delta'$ *iff* $q \xrightarrow{!?x} q'$,
- $\big[[q, B], \tau, [q', B]\big] \in \delta'$ *iff* $q \xrightarrow{\tau} q'$,
- $\big[[q, B], !x, [q', B + [x]]\big] \in \delta'$ *iff* $q \xrightarrow{!x} q'$ *and* $B(x) < k$, *and*

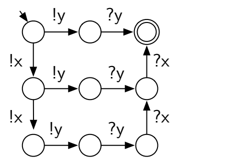
– $[[q, B + [x]], ?x, [q', B]] \in \delta'$ *iff* $q \xrightarrow{?x} q'$.

*Thereby,* $Bags_k(M_A)$ *denotes the set of multisets such that* $m \in Bags_k(M_A)$ *implies* $m(x) \leq k$ *for all* $x \in M_A$.

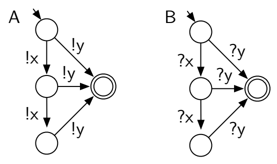
The bound $k$ can also be used as a parameter for realizability:

**Definition 16 (*k*-realizability).** *Let $C$ be a choreography automaton and* $k \in \mathbb{N}$. *$C$ is (completely/distributedly/partially) $k$-*realizable *iff $C_k$ is (completely/distributedly/partially) realizable.*

Figure 8(a) shows the unbounded choreography from Fig. 2(c) is transformed into a 2-bounded choreography which is completely 2-realizable, cf. Fig. 8(b).



(a) 2-bounded choreography



(b) completely 2-realizing peers

**Fig. 8.** Enforcing a message bound to achieve 2-realizability.

## 6 Combining Choreographies and Local Models

There are two approaches to model a choreography. The first approach focuses on the interaction between services and uses message exchange events as basic building blocks. These *interaction models* have already been discussed in Sect. 2. Interaction models are a means to quickly specify a choreography by only modeling the desired observable behavior instead of the local control flow of each peer. With the notion of realizability, these missing local behaviors can then be derived from the choreography. To this end, interaction models are best suited if all peer implementations are unknown. Interaction modeling follows a top-down approach from an abstract global model to concrete peer implementations.

In contrast, the second approach is to specify the choreography implicitly by providing a set of peer implementation and information on their interconnection. As these *interconnected models* specify both the local behavior of the participating services and their interaction, they are close to implementation. Examples of specification languages that follow this modeling style are BPMN and BPEL4Chor [18]. Interconnected models aim at reusing existing peers in new settings. Though first approaches exist to synthesize individual peers [19], this modeling style can only be used in a late stage of development.

However, a setting in which the local behaviors of some peers are completely specified whereas other peers are not specified at all is not supported by neither

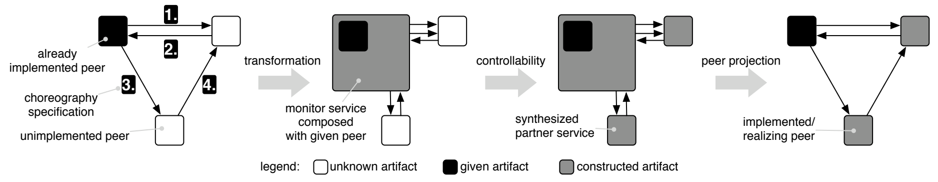


**Fig. 9.** A mixed approach to combine interaction and interconnected modeling.

modeling style. By linking realizability to controllability by transforming a choreography into an orchestration, we left the classical domain of interaction models. Instead, we derived a monitor service that orchestrates the peers. In case some peers are already implemented, they can be composed to the monitor service. This composition then specifies the remaining choreography which can be analyzed for realizability as described in Sect. 4. Figure 9 diagrams this mixed modeling style, in which choreographies and inter-organizational business processes with arbitrary levels of abstraction can be modeled. By combining both classical BPMN constructs together with iBPMN extensions [9] (i.e., modeling processes both inside and and outside pools), this mixed choreography modeling approach can be presented to modelers with a unique graphical representation.

## 7 Related Work

This is an extended version of the informal workshop paper [20]. Compared to that paper, the contributions of this paper to partner synthesis, asynchronous communication, and the combination of interaction and interconnected models are original.

Realizability received much attention in recent literature, and was studied for most of the aforementioned interaction modeling languages, see [1] for a survey. Beside the different specification languages, the approaches differ in (i) the expressiveness of the specification language (the main differences concern the support of arbitrary looping) and (ii) the nature of the message exchange (synchronous vs. asynchronous) of the realizing peers. In the following, we classify related approaches into these two groups.

*Structural restrictions.* Alur et al. [3] present necessary and sufficient criteria to realize a choreography specified by a set of message sequence charts (MSCs) with a set of concurrent automata. Both synchronous and asynchronous communication is supported. Their proposed algorithms are very efficient, but are limited to acyclic choreography specifications since the MSC model used in the paper does not support arbitrary iteration which excludes models such as Fig. 6(c).

In [21], complete and partial realizability of choreographies specified by collaboration diagrams is investigated. The authors express the realizability problem in terms of LOTOS and present a case study conducted with a LOTOS verification tool. Their approach tackle both synchronous and asynchronous communication (using bounded FIFO queues). Collaboration diagrams, however, have only limited support for repetitive behavior (only single events can be iterated and cycles such as in Fig. 6(c) cannot be expressed) and choices (events can be skipped, but complex decisions cannot be modeled). These restrictions also apply to [8] in which sufficient conditions for complete realizability of collaboration diagrams are elaborated.

*Communication models.* Realizability of conversation protocols by asynchronously communicating Büchi automata is examined in [2]. The authors show decidability of the problem and define a necessary condition for complete realizability. One of the prerequisites, *synchronous compatibility*, heavily restricts asynchronous communication.

Algorithms to check choreographies for partial realizability are discussed in [10]. Both the global and local model are specified in Let's Dance and only atomic message exchanges considered. Decker and Weske [6] study realizability of interaction Petri nets. To the best of our knowledge, it is the only approach in which (complete and partial) realizability is not defined in terms of complete trace equivalence (cf. Def. 5). Instead, the authors require the peer implementations and the choreography to be branching bisimilar. Message exchange specified by interaction Petri nets is, however, inherently synchronous.

Kazhamiakin and Pistore [22] study a variety of communication models and their impact on realizability. They provide an algorithm that finds the "simplest" communication model under which a given choreography can be completely realized. Their approach is limited to complete realizability and gives no diagnosis information in case the choreography cannot be implemented by peers. Furthermore, they fix the communication model for all messages instead of allowing different communication models for each message.

The original contribution of this paper is an automaton framework to specify arbitrary regular choreographies, check for various realizability notations (Thm. 1 states necessary *and* sufficient criteria), and to synthesize peer services that implement as much behavior as possible. Thereby, it is possible to define the message model individually for each message. Additionally, the defined synthesis algorithm provides diagnosis information that can help to fix choreographies towards complete realizability.

## 8 Conclusion

In this paper, we linked the realizability problem of choreographies to the controllability problem of orchestrations. The close relationship between these problems offers a uniform way to analyze and model arbitrary interacting services. By transforming a choreography specification into a service orchestration, we were able to reuse techniques that were originally proposed to check for controllability. These techniques resulted in a formal framework that allows to specify and analyse choreographies with both synchronous and asynchronous communication. In addition, we refined the existing hierarchy of realizability notions by defining the novel notion of distributed realizability. Finally, we proposed to combine interaction models and interconnected models.

In future work, further consequences of the relationship between controllability and realizability need to be examined. For instance, controllability is used in a number of applications such as test case generation [23] or service mediation [24]. We expect these techniques to be similarly applicable to choreographies.

**Acknowledgements** This work is funded by the DFG project "Operating Guidelines for Services" (WO 1466/8-1).

## References

1. Su, J., Bultan, T., Fu, X., Zhao, X.: Towards a theory of Web service choreographies. In: WS-FM 2007. LNCS 4937, Springer (2008) 1–16
2. Fu, X., Bultan, T., Su, J.: Conversation protocols: a formalism for specification and verification of reactive electronic services. Theor. Comput. Sci. **328**(1-2) (2004) 19–37
3. Alur, R., Etessami, K., Yannakakis, M.: Inference of message sequence charts. IEEE Trans. Software Eng. **29**(7) (2003) 623–633
4. Wolf, K.: Does my service have partners? LNCS ToPNoC **II**(5460) (2009) 152–171
5. Brand, D., Zafiropulo, P.: On communicating finite-state machines. J. ACM **30**(2) (1983) 323–342
6. Decker, G., Weske, M.: Local enforceability in interaction Petri nets. In: BPM 2007. LNCS 4714, Springer (2007) 305–319
7. Zaha, J.M., Barros, A.P., Dumas, M., Hofstede, A.H.M.t.: Let's dance: A language for service behavior modeling. In: OTM 2006. LNCS 4275, Springer (2006) 145–162
8. Bultan, T., Fu, X.: Specification of realizable service conversations using collaboration diagrams. SOCA **2**(1) (2008) 27–39
9. Decker, G., Barros, A.P.: Interaction modeling using BPMN. In: BPM Workshops 2007. LNCS 4928, Springer (2007) 208–219
10. Zaha, J.M., Dumas, M., Hofstede, A.H.M.t., Barros, A.P., Decker, G.: Service interaction modeling: Bridging global and local views. In: EDOC 2006, IEEE Computer Society (2006) 45–55
11. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting BPEL processes. In: BPM 2006. LNCS 4102, Springer (2006) 17–32
12. Leymann, F.: The (service) bus: Services penetrate everyday life. In: ICSOC 2005. LNCS 3826, Springer (2005) 12–20
13. Kopp, O., Lessen, T.v., Nitzsche, J.: The need for a choreography-aware service bus. In: YR-SOC 2008, Imperial College, London, UK (2008) 28–34
14. Decker, G.: Realizability of interaction models. In: ZEUS 2009. CEUR Workshop Proceedings Vol. 438, CEUR-WS.org (2009) 55–60
15. Decker, G., Barros, A., Kraft, F.M., Lohmann, N.: Non-desynchronizable service choreographies. In: ICSOC 2008. LNCS 5364, Springer (2008) 331–346
16. Fu, X., Bultan, T., Su, J.: Synchronizability of conversations among Web services. IEEE Trans. Software Eng. **31**(12) (2005) 1042–1055
17. 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) (2008) 374–378

18. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for modeling choreographies. In: ICWS 2007, IEEE Computer Society (2007) 296–303
19. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and participant synthesis. In: WS-FM 2007. LNCS 4937, Springer (2008) 46–60
20. Lohmann, N., Wolf, K.: Realizability is controllability. In: ZEUS 2009. CEUR Workshop Proceedings Vol. 438, CEUR-WS.org (2009) 61–67
21. Salaün, G., Bultan, T.: Realizability of choreographies using process algebra encodings. In: IFM 2009. LNCS 5423, Springer (2009) 167–182
22. Kazhamiakin, R., Pistore, M.: Analysis of realizability conditions for Web service choreographies. In: FORTE 2006. LNCS 4229, Springer (2006) 61–76
23. Kaschner, K., Lohmann, N.: Automatic test case generation for interacting services. In: ICSOC 2008 Workshops. LNCS 5472, Springer (2009) 66–78
24. Gierds, C., Mooij, A.J., Wolf, K.: Specifying and generating behavioral service adapter based on transformation rules. Preprint CS-02-08, Universität Rostock, Rostock, Germany (2008) (submitted to a journal).