Automated Choreography Repair*

Samik Basu[1], Tevfik Bultan[2]

[1] Iowa State University, sbasu@iastate.edu
[2] University of California at Santa Barbara, bultan@cs.ucsb.edu

Abstract. Choreography analysis is a crucial problem in concurrent and distributed system development. A choreography specifies the desired ordering of message exchanges among the components of a system. The realizability of a choreography amounts to determining the existence of components whose communication behavior conforms to the given choreography. The realizability problem has been shown to be decidable. In this paper, we investigate the repairability of un-realizable choreographies, where the goal is to identify a set of changes to a given un-realizable choreography that will make it realizable. We present a technique for automatically repairing un-realizable choreographies and provide formal guarantees of correctness and termination. We demonstrate the viability of our technique by applying it to several representative unrealizable choreographies from Singulary OS channel contracts and Web services.

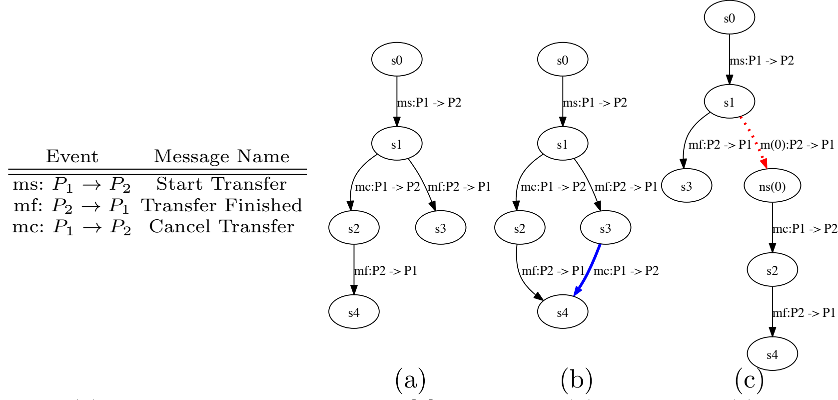
1 Introduction

Choreography specifications are used in a variety of domains including coordination of software in service-oriented computing [18], specification of process interactions in Singularity OS [11], and specification of communication behavior among processes in distributed programs [2]. Choreographies describe desired message exchange sequences among components, programs or processes (we will refer to them as *peers*) of a distributed system. The choreography realizability problem is determining whether one can construct peers whose interaction behavior conforms to the given choreography. As an example, consider the choreography over two peers P_1 and P_2 shown in Figure 1(a) where edges represent messages sent from one peer to another. This choreography describes a simple file transfer protocol [9] where P_1 is the client asking for the file transfer and P_2 is the file server. First, the client sends a message to the server to request that the server starts the transfer. When the transfer is finished, the server sends the "Transfer Finished" message and the protocol terminates. However, the client may decide to cancel the transfer before hearing back from the server by sending a "Cancel Transfer" message in which case the server responds with "Transfer Finished" message, which, again, terminates the protocol.

Figure 3(a) presents the projection of the choreography onto the participating peers resulting in the corresponding peer behaviors (send actions are denoted by "!" and receive actions are denoted by "?"). The distributed system that consists of the peer specifications shown in Figure 3(a) can generate the message sequence:

$$ms^{P_1 \to P_2}, mf^{P_2 \to P_1}, mc^{P_1 \to P_2} \quad (1)$$

* This work is partially supported by NSF grants CCF 1155780 and CCF117708.



Event	Message Name
ms: $P_1 \rightarrow P_2$	Start Transfer
mf: $P_2 \rightarrow P_1$	Transfer Finished
mc: $P_1 \rightarrow P_2$	Cancel Transfer

Fig. 1. (a) Un-realizable choreography [9]; Repair by (b) relaxation, (c) restriction.

This sequence corresponds to the case where the server sends a "Transfer Finished" message (mf), but before consuming that message, the client sends the cancellation request message (mc). The sequence moves the server to an undefined (error) configuration, where the server does not know whether the file was transferred completely to the client before the client sent the cancellation request. In terms of the choreography specification shown in Figure 1(a), the message sequence given above is not covered by the choreography, but any implementation of this choreography that uses asynchronous message passing will generate the message sequence: (1), violating the choreography specification. Hence, the choreography specification shown in Figure 1(a) is un-realizable.

Problem Statement. This brings up the question: *when a choreography is determined to be un-realizable, is it possible to automatically repair the choreography such that the repaired version is realizable?* We will refer to this problem as the *choreography repairability problem*. Its importance stems from the fact that automation in repairing choreography will allow faster development of distributed systems with formal guarantees of correctness.

Our Solution. Our choreography repair technique analyzes and eliminates the cause of violation of the condition for choreography realizability. In [4], we have proved that choreography $\mathcal{C}$ is realizable if and only if its behavior (i.e., the set of message sequences generated by $\mathcal{C}$, denoted as $\mathcal{L}(\mathcal{C})$) is identical to the behavior of $\mathcal{I}_1^{\mathcal{C}}$ (denoted as $\mathcal{L}(\mathcal{I}_1^{\mathcal{C}})$), where $\mathcal{I}_1^{\mathcal{C}}$ is the asynchronous system in which each participating peer has at most one pending message at any point of time, and is obtained from the projection of $\mathcal{C}$. We present two types of choreography repair mechanisms that eliminate the differences between $\mathcal{L}(\mathcal{C})$ and $\mathcal{L}(\mathcal{I}_1^{\mathcal{C}})$:

1. *Relaxation.* The choreography $\mathcal{C}$ is changed to $\mathcal{C}'$ such that $\mathcal{L}(\mathcal{C}) \subseteq \mathcal{L}(\mathcal{C}')$, i.e., new behavior is added to $\mathcal{C}$, such that $\mathcal{L}(\mathcal{C}') = \mathcal{L}(\mathcal{I}_1^{\mathcal{C}'})$.
2. *Restriction.* The choreography $\mathcal{C}$ is changed to $\mathcal{C}'$ such that $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{C}' \downarrow_{\mathcal{C}}) = \mathcal{L}(\mathcal{I}_1^{\mathcal{C}'} \downarrow_{\mathcal{C}}) \subseteq \mathcal{L}(\mathcal{I}_1^{\mathcal{C}})$, where $\downarrow_{\mathcal{C}}$ denotes the behavior projected on the messages in $\mathcal{C}$. This change implies that some behavior of $\mathcal{I}_1^{\mathcal{C}}$ is disallowed in $\mathcal{I}_1^{\mathcal{C}'}$. This is achieved by adding extra synchronization messages in $\mathcal{C}'$. When these extra

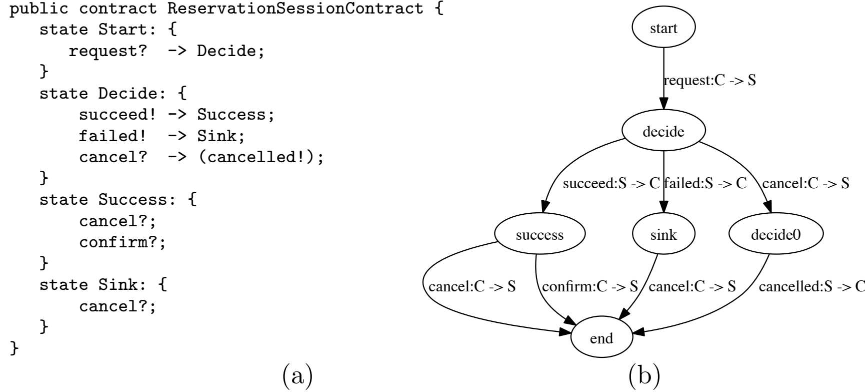


Fig. 2. (a) Channel Contract for ReservationSession and (b) the corresponding state machine

messages are projected away, the repaired choreography $\mathcal{C}'$ specifies exactly the same sequences of messages specified by the un-realizable choreography $\mathcal{C}$.

For example, the choreography in Figure 1(a) is changed to the one in Figure 1(b) via relaxation, by adding new behavior (blue bold-edge), which makes the latter realizable. This is because the sequence that made $\mathcal{C}$ un-realizable (see sequence (1) above) is now included in the repaired version $\mathcal{C}'$. On the other hand, Figure 1(c) demonstrates repair via restriction, by adding synchronization messages from state s_1 to $ns(0)$ (red dotted-edges); this repair also makes the resulting choreography realizable. In this case, the sequence in (1) is not possible in $\mathcal{I}_1^{\mathcal{C}'}$.

Contribution. We present a formal characterization of choreography repairability. To the best of our knowledge, this is the first time such a characterization has been presented. We present a sound and complete algorithm for choreography repair based on this characterization. We also discuss its application by demonstrating automated repair of several unrealizable choreographies. Although choreography examples we use in this paper consist of two-party choreographies, the formal model and the repair algorithm we present are general and handle multi-party choreographies.

2 Repairing Singularity OS Channel Contracts

We motivate the practical applicability of automated choreography repair using Singularity OS channel contracts. Singularity OS [16] is developed by Microsoft research with the objective of improving OS dependability by ensuring process isolation. The processes in Singularity OS communicate over FIFO channels and follow specific channel contracts (choreographies in our case); that specify allowable communication patterns between processes (client and server). The Singularity OS channel contracts correspond to choreography specifications. One problem is to determine whether one can implement a client and a server whose interaction conforms to the given channel contract, i.e., determining realizability of the given channel contract.

Figure 2(a) presents a channel contract called reservation session contract (where message declarations are omitted for brevity). The contract specifies four explicit states and the message sequences from the perspective of the server. For instance, the contract specifies that the state changes from "Start" to "Decide" when the server receives a message "request" from the client. From the state "Decide", there are three choices: the server sends the message "succeed" to the client resulting in the state update to "Success"; the server responds to the client with message "failed" leading to the state "Sink"; the client sends "cancel" followed by the server sending "cancelled" message. Figure 2(b) presents the state machine for this contract (`C` represents the client and `S` represents the server).

The Singularity Design Note 5 [16] states that the client and server processes that are verified to conform to a given channel contract (i.e., that implement the projection of the channel contract correctly) are guaranteed to interact without any deadlocks. However, in [17], the authors demonstrated that this claim is incorrect since the channel contract itself can be un-realizable, in which case the processes implemented based on the projection of the contract can deadlock. One of the examples demonstrating this problem is the reservation session contract from Singularity OS shown above. Due to asynchronous communication, the client and server can move out-of-sync and deadlock. Consider the scenario where the client sends a "cancel" message and waits for the "cancelled" message from the server, while the server sends a "failed" message and consumes the "cancel" message from the client. This sequence of interactions leads to a deadlock. In fact there are no client and server processes that can conform to this contract without deadlock while interacting via FIFO channels (as required by the Singularity OS), i.e., the choreography specified by this channel contract is un-realizable.

The automated choreography repair technique we present in this paper is directly applicable to Singularity OS channel contracts. Using our technique we can repair un-realizable channel contracts, and ensure deadlock free implementation of repaired contracts. We will discuss the application of our automated choreography repair technique to the reservation contract in Section 5.

3 Choreography Realizability

We proceed by presenting an overview of the existing results [4] on choreography realizability, which forms the basis of our automated choreography repair strategy.

Peers. The behavior $\mathcal{B}$ of a peer P is a finite state machine (M, T, t_0, δ) where M is the union of input (M^{in}) and output (M^{out}) message sets, T is the finite set of states, $t_0 \in T$ is the initial state, and $\delta \subseteq T \times (M \cup \{\epsilon\}) \times T$ is the transition relation. A transition $\tau \in \delta$ can be one of the following three types: (1) a send-transition of the form $(t_1, !m_1, t_2)$ which sends out a message $m_1 \in M^{\text{out}}$, (2) a receive-transition of the form $(t_1, ?m_2, t_2)$ which consumes a message $m_2 \in M^{\text{in}}$ from peer's input queue, and (3) an ϵ-transition of the form (t_1, ϵ, t_2). We write $t \xrightarrow{a} t'$ to denote that $(t, a, t') \in \delta$. Figure 3(a) illustrates the behavior of peers P_1 and P_2; states in P_i are denoted by a tuple (P_i:"state-name").

System. Given a set of peers $\mathcal{P} = \{P_1, \ldots, P_n\}$ with $\mathcal{B}_i = (M_i, T_i, t_{0i}, \delta_i)$ denoting the behavior of P_i and $M_i = M_i^{\text{in}} \cup M_i^{\text{out}}$ such that $\forall i : M_i^{\text{in}} \cap M_i^{\text{out}} = \emptyset$,

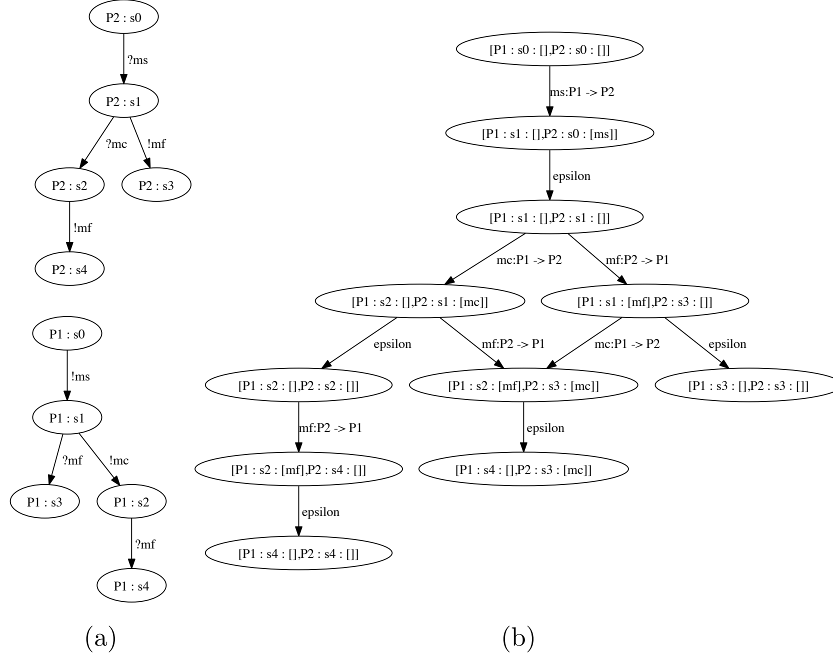


Fig. 3. (a) Projected Peers $\mathcal{P}_1$ and $\mathcal{P}_2$ for Figure 1(a); (b) System Behavior

and $\forall i, j : i \neq j \Rightarrow M_i^{\text{in}} \cap M_j^{\text{in}} = M_i^{\text{out}} \cap M_j^{\text{out}} = \emptyset$. A system behavior or simply a system over $\mathcal{P}$ is denoted by a (possibly infinite state) state machine $\mathcal{I} = (\mathcal{P}, S, s_0, M, \Delta)$ where $\mathcal{P}$ is the set of peers, S is the set of states in the system and each state $s = (Q_1, t_1, Q_2, t_2, \ldots Q_n, t_n)$ in the system is described by the local states (t_is) of the peers in $\mathcal{P}$ along with the contents of their queues (Q_is). $s_0 \in S$ is the start state, where none of the peers have any pending messages in their queue to consume. The set M contains the set of all messages that are being exchanged by the participating peers.

Finally, the transition relation Δ is described as follows. The send actions are non-blocking, i.e., when a peer P_i sends a message m to a peer P_j (denoted by $m^{P_i \rightarrow P_j}$), the message gets appended to the tail of the queue associated to P_j. We refer to the queue as the receive queue of P_j. The receive actions are blocking, i.e., a peer can only consume a message if it is present at the head of its receive queue; on consumption of the message, it is removed from the head of the queue. Only the send actions are observable in the system as these actions involve two entities: the sender sending the message and the receive queue of the receiver. All other actions are local to one peer and, therefore, unobservable (ϵ-transitions). We will use the functions $\texttt{lSt}(.,.)$ and $\texttt{lQu}(.,.)$ to obtain local state and queue of a peer from a state in the system, i.e., for $s = (Q_1, t_1, Q_2, t_2, \ldots Q_n, t_n) \in S$, $\texttt{lSt}(s, P_1) = t_1$ and $\texttt{lQu}(s, P_1) = Q_1$.

K-bounded System. A k-bounded system (denoted by $\mathcal{I}_k$) is a system where the length of message queue for any peer is at most k. In any k-bounded system,

the send actions can block if the receive queue of the receiver peer is full. Any k-bounded system is finite state as long as the behaviors of the participating peers are finite state. Figure 3(b) illustrates the system $\mathcal{I}_1$ obtained from the communicating peers P_1 and P_2 of Figure 3(a). Note that initially P_1 is at the local state $P_1\!:\!s_1$ with an empty receive queue denoted by [ ].

Choreography Specification. A choreography specification is a finite state machine $\mathcal{C} = (\mathcal{P}, S^C, s_0^C, L, \Delta^c)$ where $\mathcal{P}$ is a finite set of peers, S^C is a finite set of states, $s_0^C \in C$ is the initial state, L is a finite set of message labels and, finally, $\Delta^c \subseteq S^C \times \mathcal{P} \times L \times \mathcal{P} \times S^C$ is the transition relation. A transition of the form $(s_i^C, P, m, P', s_j^C) \in \Delta^c$ represents the sending of message m from P to P' $(P, P' \in \mathcal{P})$.

Peer Projection. The projection of a choreography $\mathcal{C}$ on one of the peers P, is obtained from $\mathcal{C}$ by performing the following updates to the state machine describing $\mathcal{C}$. (a) If a transition label is $m^{P \to P'}$ then replace it with $!m$; (b) if a transition label is $m^{P' \to P}$ then replace it with $?m$; (c) otherwise, replace transition label with ϵ. The system obtained from the asynchronous communication of the projected peers of $\mathcal{C}$ is denoted by $\mathcal{I}^{\mathcal{C}}$; $\mathcal{I}_1^{\mathcal{C}}$ being the corresponding 1-bounded system. The language of a choreography or a system is described in terms of a set of sequences of send actions of the form $m^{P \to P'}$; the concatenation of ϵ to any sequence results in the sequence itself. The language is denoted by $\mathcal{L}(.)$.

Theorem 1 (Realizability [4]). *$\mathcal{C}$ is language realizable* $\Leftrightarrow$ $[\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{I}_1^{\mathcal{C}})]$

This theorem states that a choreography is realizable if and only if the set of sequences of send actions of a choreography is identical to the set of sequences of send actions of the 1-bounded system where the participating peers are generated from the (determinized) projection of the choreography under consideration. Figure 3(b) presents the behavior of the system $\mathcal{I}_1^{\mathcal{C}}$ for the choreography specification $\mathcal{C}$ shown in Figure 1(a), where epsilon-labeled transitions denote consumption of messages and other transitions denote sending of messages. The choreography $\mathcal{C}$ is un-realizable because it does not include a specific send sequence that is possible in $\mathcal{I}_1^{\mathcal{C}}$ (Figure 3(b)) (Sequence (1) discussed in Section 1).

4 Choreography Repair

Types of Repair. In this paper, we present two alternative techniques for repairing un-realizable choreographies. One is based on adding new behaviors (in terms of sends) to $\mathcal{C}$, which we call *relaxation*. The other is based on adding constraints that do not alter allowed sequences of sends in $\mathcal{C}$ but restrict the behavior in $\mathcal{I}_1^{\mathcal{C}}$. We call this approach *restriction*. The techniques will be based on the observation that from Theorem 1 and from the nature of asynchrony, it follows: $\mathcal{L}(\mathcal{C}) \neq \mathcal{L}(\mathcal{I}_1^{\mathcal{C}}) \;\Rightarrow\; \mathcal{L}(\mathcal{C}) \subset \mathcal{L}(\mathcal{I}_1^{\mathcal{C}})$.

State Relationships between $\mathcal{I}_1^{\mathcal{C}}$ and $\mathcal{C}$. Before we describe the repair techniques, we first discuss the structure of the $\mathcal{I}_1^{\mathcal{C}}$, which is crucial for understanding our approach. If a state in $\mathcal{C}$ is represented as $s^{\mathcal{C}}$, then the corresponding state in the peer P is a tuple denoted by $P\!:\!s^{\mathcal{C}}$. Proceeding further, if s is a state in $\mathcal{I}_1^{\mathcal{C}}$, then $s = (Q_1, t_1, \ldots, Q_n, t_n)$, where n is the number of peers and t_i is of the

form $P_i\!:\!s_i^{\mathcal{C}}$. Note that, the local states of each peer in s may have been obtained from different states $s_i^{\mathcal{C}}$ in $\mathcal{C}$.

Consider for example, the second state of the system in Figure 3(b)–P_1 is at a state $P_1\!:\!s_1$ obtained from the state s_1 in $\mathcal{C}$ and P_2 is at a state $P_2\!:\!s_0$ obtained from the state s_0 in $\mathcal{C}$. Using the notations introduced in Section 3, $\texttt{lSt}((P_1\!:\!s_1\!:\![\,], P_2\!:\!s_0\!:\![ms]), P_1) = P_1\!:\!s_1$; $\texttt{lQu}((P_1\!:\!s_1\!:\![\,], P_2\!:\!s_0\!:\![ms]), P_2) = [ms]$.

4.1 Differences between $\mathcal{C}$ and $\mathcal{I}_1^{\mathcal{C}}$

In order to apply relaxation or restriction, it is important to identify at least one difference between $\mathcal{C}$ and $\mathcal{I}_1^{\mathcal{C}}$ in terms of sequences of send actions. We know that for un-realizable $\mathcal{C}$, $\mathcal{L}(\mathcal{C}) \subset \mathcal{L}(\mathcal{I}_1^{\mathcal{C}})$. Therefore, there exists at least one send sequence in $\mathcal{I}_1^{\mathcal{C}}$ which is absent in $\mathcal{C}$.

Consider that there exists a path in $\mathcal{I}_1^{\mathcal{C}}$ in the form

$$s_1 \xrightarrow{m_1^{P_1\to P_1'}} s_2 \xrightarrow{m_2^{P_2\to P_2'}} s_3 \to \ldots s_i \xrightarrow{m_i^{P_i\to P_i'}} s_{i+1} \tag{2}$$

which generates the following sequence of send actions $m_1^{P_1\to P_1'}, m_2^{P_2\to P_2'}, \ldots,$ $m_i^{P_i\to P_i'}$. Assume that, none of the paths in $\mathcal{C}$ allow the above send sequence. However, there exists a path in $\mathcal{C}$ which replicates the above sequence till $m_{i-1}^{P_{i-1}\to P_{i-1}'}$. Let such a path be denoted by

$$t_1 \xrightarrow{m_1^{P_1\to P_1'}} t_2 \xrightarrow{m_2^{P_2\to P_2'}} t_3 \to \ldots t_{i-1} \xrightarrow{m_{i-1}^{P_{i-1}\to P_{i-1}'}} t_i \tag{3}$$

where t_i does not have any outgoing transition labeled by $m_i^{P_i\to P_i'}$. In summary, one of the differences between the send sequences present in $\mathcal{C}$ and $\mathcal{I}_1^{\mathcal{C}}$ is due to the presence of send action $m_i^{P_i\to P_i'}$ at s_i and absence of the same at t_i. For instance, going back to the example in Figure 3, the difference between $\mathcal{C}$ and $\mathcal{I}_1^{\mathcal{C}}$ is due to $ms^{P_1\to P_2}, mf^{P_2\to P_1}, mc^{P_1\to P_2}$, in which case s_i is equal to $(P_1\!:\!s_1\!:\![mf], P_2\!:\!s_3\!:\![])$ in $\mathcal{I}_1^{\mathcal{C}}$ and t_i is equal to s_3 in $\mathcal{C}$. The cause of the difference between the behaviors can be explained in one of the two ways:

Independent Branches. The choreography specification includes a branching behavior involving sends from at least two peers in two different branches. The sender peers follow different paths in the branches. This is the case in Figure 1(a).

Independent Sequences. The choreography specification includes a path where there are two messages sent by two different peers and the sender of the second message does not depend on the first message. This situation can be illustrated using the following choreography specification: $t_0 \xrightarrow{m^{P_1\to P_2}} t_1 \xrightarrow{m^{P_3\to P_4}} t_2$. The first and second transitions correspond to send actions of P_1 and P_3, which can occur in any order in the corresponding system and therefore, this choreography, therefore, cannot be realized. We will refer to the path as independent sequences and the transitions as *independent transitions*.

The objective of repair via relaxation or restriction is to alter the behavior of $\mathcal{C}$ proceeding from t_i such that the above causes of differences can be eliminated.

4.2 Repair by Relaxation

As noted before, relaxing $\mathcal{C}$ corresponds to adding new behaviors to $\mathcal{C}$. Specifically, adding a new behavior from state t_i (in path (3) above) implies adding a transition

from t_i to some t'_i with transition label $m_i^{P_i \rightarrow P'_i}$. The addition of such a new transition obviously results in a new choreography specification, say $\mathcal{C}'$. We will denote relaxation of $\mathcal{C}$ to $\mathcal{C}'$ as $\mathcal{C} \nearrow \mathcal{C}'$. Note that, the following holds: $\mathcal{C} \nearrow \mathcal{C}' \Rightarrow \mathcal{L}(\mathcal{C}) \subseteq \mathcal{L}(\mathcal{C}')$.

While adding a new transition from t_i to a state (say t'_i) eliminates the difference due to the send action $m_i^{P_i \rightarrow P'_i}$, the important next step is to identify a suitable t'_i. There are two possibilities: we can either assign t'_i to some existing state in $\mathcal{C}$ or generate a new state. Careful selection of one of the two choices is important because it impacts the termination of the repair mechanism. Using the form of the system path shown in (2), let $\mathtt{lSt}(s_i, P_i) = P_i : c_i$; $\mathtt{lSt}(s_{i+1}, P_i) = P_i : c_{i+1}$; $\mathtt{lQu}(s_i, P_i) = Q_i$; $\mathtt{lQu}(s_{i+1}, P_i) = Q_{i+1}$. In the above, $Q_i = Q_{i+1}$ because the peer P_i does not consume any messages at this transition.

Case 1. Consider that the receive queue Q_i of the peer P_i is non-empty, implying that there is one pending message to be consumed (recall that the $\mathcal{I}_1^{\mathcal{C}}$ is 1-bounded system with each receive queue capacity being 1). In other words, some peer (say, R) has sent the message (say m) to P_i and P_i has not encountered any receive action along the choreography path it has taken resulting in system path shown in (2).

This case corresponds to the situation described as *independent branching* (see above), when peer P_i is moving along a choreography specification path π and the other peer R is moving along a different path π' of the choreography specification, resulting in the path shown in (2). Furthermore, R has sent m to P_i which resides un-consumed in the receive queue of P_i.

Case 1a. Let there be a transition in the behavior of peer P_i at state $P_i : c_{i+1}$, where it can consume the message in its queue: $P_i : c_{i+1} \xrightarrow{?m} P_i : c'_i$. That is, the choreography specification includes $c_{i+1} \xrightarrow{m^{R \rightarrow P_i}} c'_i$ along the path π. Therefore, both of the paths under consideration, π and π', have the send action $m^{R \rightarrow P_i}$. In π, $m_i^{P_i \rightarrow P'_i}$ is followed by $m^{R \rightarrow P_i}$. In π', $m^{R \rightarrow P_i}$ is not followed by $m_i^{P_i \rightarrow P'_i}$.

In this case, the relaxation adds $t_i \xrightarrow{m_i^{P_i \rightarrow P'_i}} t'_i$ in the choreography specification and sets t'_i to c'_i.

Case 1b. On the other hand, if there exists no transition in the behavior of peer P_i starting from state $P_i : c_{i+1}$ where it can consume the message in its queue, then the following repair is done.

Case 1b-i. If $P_i : c_{i+1}$ belongs to a cycle then in the newly added transition $t_i \xrightarrow{m_i^{P_i \rightarrow P'_i}} t'_i$, t'_i is set to a newly generated state, which replicates the choreography specification starting from c_{i+1}. Note that, the repair does not assign t'_i to c_{i+1}. This is because such assignment will result in unnecessary over-relaxation of choreography specification due to the presence in $m^{R \rightarrow P_i}$ in path π' and its possible absence in the cycle which is part of the path π. We will discuss below this scenario using the example in Figure 4.

Case 1b-ii. If P_i at $P_i : c_{i+1}$ cannot consume the pending message and $P_i : c_{i+1}$ does not belong to any cycle, then t'_i is set to a newly generated state. The

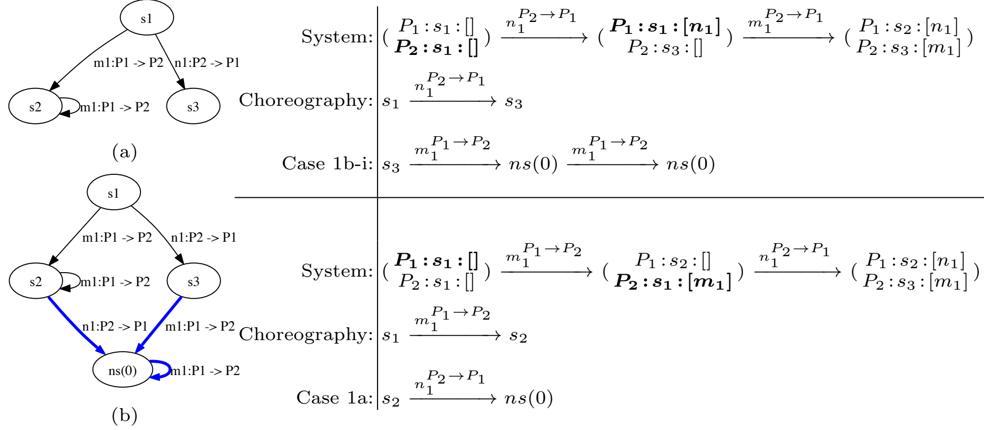


Fig. 4. Example illustrating application of Case 1b-ii and 1a of relaxation

addition of the new transition removes the identified difference between the choreography and the system.

For instance, in Figure 3(b), the path in $\mathcal{I}_1^{\mathcal{C}}$ that is absent in $\mathcal{C}$ (Figure 1(a)) has the sequence $ms^{P_1 \to P_2}$, $mf^{P_2 \to P_1}$, $mc^{P_1 \to P_2}$. Note that, we are considering only the send actions and the transitions are considered with zero or more occurrences of ϵ followed by a send action. The path in $\mathcal{C}$ that replicates most of this sequence is $s_0 \xrightarrow{ms^{P_1 \to P_2}} s_1 \xrightarrow{mf^{P_2 \to P_1}} s_3$. Therefore, for repair by relaxation, our objective is to add a transition with send action $mc^{P_1 \to P_2}$ from the choreography state s_3. From the system, we know that the peer P_1 at the state $P_1 : s_2$ can consume the message m_f in its receive queue and move to a state in $P_1 : s_4$ (see Figure 3). Therefore, the transition added from s_3 has the destination state s_4. The result of this repair by relaxation is the choreography specification presented in Figure 1(b). This illustrates the **Case 1a** of repair by relaxation.

Figure 4 illustrates the applications of **Case 1b-i** and **1a**. The local states of the peers participating in the system transitions are presented in bold-font. In the first step, the difference between the system transition sequence and the choreography sequence is repaired following the Case 1b-i. $P_1 : s_2$ does not have a transition where it consumes the pending message n_1, and $P_1 : s_2$ belongs to a cycle. Therefore, a new state $ns(0)$ replicating s_2 is generated as part of the repair strategy instead of adding the transition $m_1^{P_1 \to P_2}$ from s_3 to s_2.

Case 2. Now consider that the receive queue Q_i of the peer P_i is empty, implying that there is no pending message to be consumed. Unlike the previous case, in this situation, the difference between $\mathcal{I}_1^{\mathcal{C}}$ and $\mathcal{C}$ (represented by paths (2) and (3) in Section 4.1) is *not necessarily* due to independent branches, when two peers move along two different paths of the choreography specification.

Instead the peers may be moving along the same path of the choreography specification, and the latter has imposed an "un-realizable" ordering of send actions involving $m_i^{P_i \to P'_i}$. In other words, it is not possible to "stop" P_i from sending the message m_i from its projected behavior when the choreography

specification reaches t_i, however t_i does not have $m_i^{P_i \rightarrow P'_i}$. This corresponds to the case of independent sequences (see above).

Recall that, the choreography specification state is t_i from where there is no matching $m_i^{P_i \rightarrow P'_i}$ event. We check whether there exists a path from $P_i : t_i$ (i.e., local state of P_i obtained from projection at t_i) to $P_i : t_i$ in the peer P_i via a sequence of transitions such that after a sequence of ϵ-transitions, there is a $!m_i$ transition followed by some other sequence of transitions.

Case 2a. If the check is successful, then we can infer that t_i is part of a loop and it contains independent transitions, which cause un-realizability.

Case 2a-i. Then we identify the first intermediate state $P_i : t$ in this loop, which has an outgoing transition over some other output action. In this case, a new transition $t_i \xrightarrow{m_i^{P_i \rightarrow P'_i}} t'_i$ with t'_i set to t is added to replicate the behavior in $\mathcal{I}_1^{\mathcal{C}}$.

Case 2a-ii. If no such intermediate state exists, then $t_i \xrightarrow{m_i^{P_i \rightarrow P'_i}} t'_i$ with t'_i set to t_i (self-loop) is added.

In either case, the permutations of pairs of independent transitions that were identified as the difference between $\mathcal{C}$ and $\mathcal{I}_1^{\mathcal{C}}$ are added and nothing else.

Case 2b. On the other hand, if the check is unsuccessful, then we can infer that t_i is not part of a loop.

Case 2b-i. We find out whether $P : c_{i+1}$ (local state of the sender at s_{i+1}) has a path to $P : t_i$ (t_i being the choreography state that cannot replicate the behavior of the system from s_i). If such path exists in the behavior of P_i, we infer that P_i moves along a path different from $t_1, t_2, \ldots, t_i$ (see path 3) in choreography but the path has the ability to join at t_i. In this case, we add a new transition labeled with $t_i \xrightarrow{m_i^{P_i \rightarrow P'_i}} c_{i+1}$ to remove the difference between the choreography and corresponding the system.

Case 2b-ii. If the condition in Case 2b-i fails, then we find out the choreography state reachable from c_{i+1} (the choreography state corresponding the senders local state at s_{i+1}) via the action $m_{i-1}^{P_{i-1} \rightarrow P'_{i-1}}$. If such a state is t, then this implies that the choreography path extending from c_{i+1} allows $m_{i-1}^{P_{i-1} \rightarrow P'_{i-1}}$ after $m_i^{P_i \rightarrow P'_i}$, while the choreography path along $t_1, t_2, \ldots, t_i$ (see path 3) does not allow $m_i^{P_i \rightarrow P'_i}$ after $m_{i-1}^{P_{i-1} \rightarrow P'_{i-1}}$. The repair in this case is similar to Case 1a and amounts to adding $t_i \xrightarrow{m_i^{P_i \rightarrow P'_i}} t$. On the other hand, if no such choreography state t exists, then a new state is generated and a transition over $m_i^{P_i \rightarrow P'_i}$ is added from t_i to this newly generated state.

Figure 5 illustrates the application of Case 2 of relaxation.

4.3 Repair by Restriction

The objective of restriction, unlike relaxation, is to constrain the behavior of the system $\mathcal{I}_1^{\mathcal{C}}$. In other words, going back to paths (3) and (2) in Section 4.1,

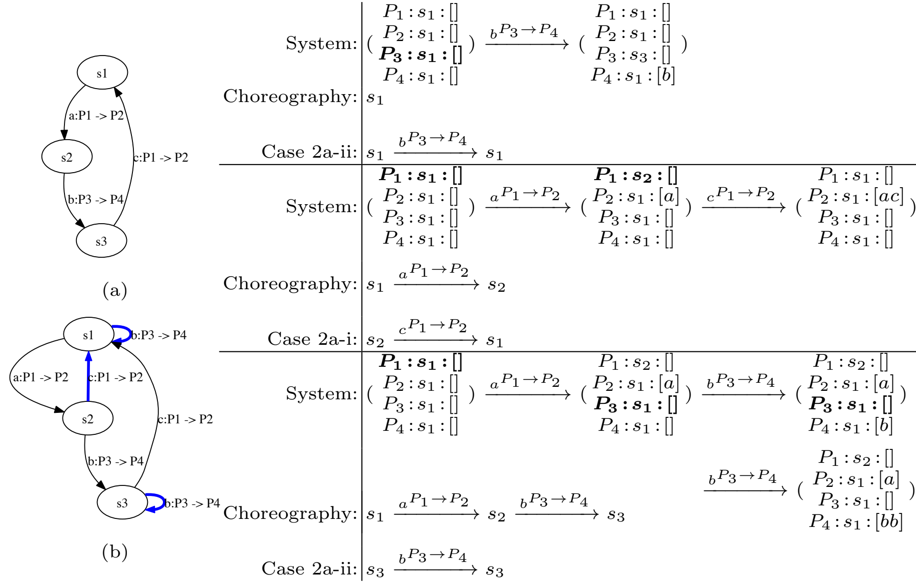


Fig. 5. Example illustrating application of Case 2a of Relaxation

restriction implies disallowing the transition $s_i \xrightarrow{m_i^{P_i \to P_i'}} s_{i+1}$ in $\mathcal{I}_1^{\mathcal{C}}$ i.e., introducing restriction to disallow the transition $c_i \xrightarrow{m_i^{P_i \to P_i'}} c_i'$ in $\mathcal{C}$ from happening at the system state s_i, where $\texttt{lSt}(s_i, P_i) = P_i : c_i$ and $\texttt{lSt}(s_{i+1}, P_i) = P_i : c_{i+1}$. The restriction of transition $c_i \xrightarrow{m_i^{P_i \to P_i'}} c_i'$ is achieved by adding a new intermediate state between c_i and c_i'.

Case 1. Let t_i have a transition to t where some peer P sends a message m to P' and P is different from P_i, the sender peer of the message m_i. We verify whether the transition $c_i \xrightarrow{m_i^{P_i \to P_i'}} c_{i+1}$ is reachable from t.

If the verification is successful, this corresponds to the case of unrealizability due to *independent transitions*. The repair, in this case, results from the addition of an intermediate state between t_i and t such that $t_i \xrightarrow{m^{P \to P'}} ns \xrightarrow{nm^{P' \to P_i}} t$, where nm is a new message and ns is a new state. Addition of such transitions will disallow the $m_i^{P_i \to P_i'}$ at the system state s_i.

Case 2. However, if there is no transition from the state t_i or the transition is labeled with a send action performed by the same peer P_i, then it corresponds to the case of unrealizability due to *independent branches*. In this case, we identify the sender peer P_{i-1} for the transition from t_{i-1} to t_i. The restriction is achieved by introducing an intermediate state between c_i and c_{i+1} as follows: $c_i \xrightarrow{nm^{P_{i-1} \to P_i}} ns \xrightarrow{m_i^{P_i \to P_i'}} c_{i+1}$, where nm and ns are newly added message and newly added state, respectively.

Algorithm 1 Repair($\mathcal{C}$, inputRepairMechanism)

1: Compute $\mathcal{I}_1^{\mathcal{C}}$
2: **if** $\mathcal{L}(\mathcal{C}) = \mathcal{L}(\mathcal{I}_1^{\mathcal{C}})$ return $\mathcal{C}$ ▷ $\mathcal{C}$ is realizable
3: Find a difference between $\mathcal{C}$ and $\mathcal{I}_1^{\mathcal{C}}$ ▷ Sec.4.1
4: Apply $\mathcal{C}$ inputRepairMechanism $\mathcal{C}'$ ▷ Sec.4.2, 4.3
5: GOTO Line 1 with $\mathcal{C}$ assigned to $\mathcal{C}'$ ▷ Iterate

These newly added messages and transitions in the choreography can be viewed as an extra step which forces the peer P_i to come in sync with some other peer (P' in Case 1a above and P in Case 1b and 2 above) before sending the message m_i. We refer to such extra step as the *synchronization step.*

We will denote restriction of $\mathcal{C}$ to generate $\mathcal{C}'$ as $\mathcal{C} \searrow \mathcal{C}'$. It is immediate that

$$\mathcal{C} \searrow \mathcal{C}' \;\Rightarrow\; \mathcal{L}(\mathcal{C}' \downarrow_{\mathcal{C}}) = \mathcal{L}(\mathcal{C}') \;\wedge\; \mathcal{L}(\mathcal{I}_1^{\mathcal{C}'} \downarrow_{\mathcal{C}}) \subseteq \mathcal{L}(\mathcal{I}_1^{\mathcal{C}}) \quad (4)$$

The operation $'.' \downarrow_{\mathcal{C}}$ extracts the behavior with respect to actions present in $\mathcal{C}$. The restriction does not alter the behavior of the choreography in terms of the actions in $\mathcal{C}$ but restricts the behavior of the corresponding system in terms of the actions in $\mathcal{C}$. Figure 1(c) presents the result of applying restriction based repair of the choreography in Figure 1(a). There exists a path in the system where it reaches the state $P_1 : s_1 : [mf], P_2 : s_3 : []$ via the send sequence $ms^{P_1 \to P_2}$, $mf^{P_2 \to P_1}$; from this state, the system is capable of producing $mc^{P_1 \to P_2}$ (see Figure 3). The choreography via the same sequence of sends reaches the state s_3. Therefore, the restriction is achieved by following the Case 2 above resulting in a repaired choreography in Figure 1(c).

4.4 Iterative Algorithm

It is necessary to apply the relaxation or the restriction iteratively till a realizable choreography is obtained and all differences between the choreography and the corresponding 1-bounded system behavior have been resolved. In Algorithm 1 the input parameter "inputRepairMechanism" is either set to $\nearrow$ (relaxation) or $\searrow$ (restriction). Figures 4 and 5 illustrate the application of Algorithm 1.

Theorem 2 (Correctness). *The algorithm* Repair *is guaranteed to terminate and produce a repaired (i.e., realizable) choreography.*

Proof Sketch. The algorithm iterates as long as there is a difference between the choreography $\mathcal{C}$ and the interaction behavior of the corresponding system $\mathcal{I}_1^{\mathcal{C}}$. To address the difference, the algorithm introduces new states as part of the repair process. The number of such introduction of new states depends directly on the number of independent branches and independent transitions (that cause un-realizability of the choreography). The number of independencies are bounded by the number of branches and the maximum length of a path (with one unfolding) in the choreography, which ensures the boundedness in the introduction of new states. This, in turn, ensures that all possible causes of choreography un-realizability is removed within finite number of steps. □

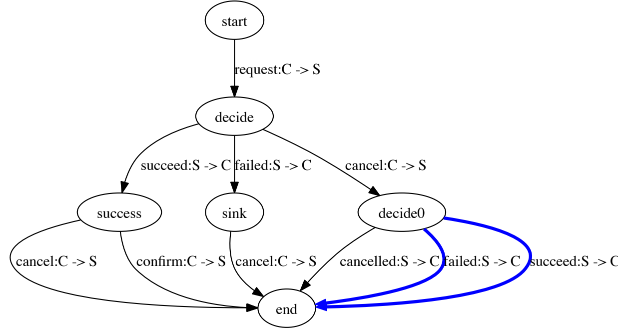


Fig. 6. ReservationSession Contract repaired by relaxation

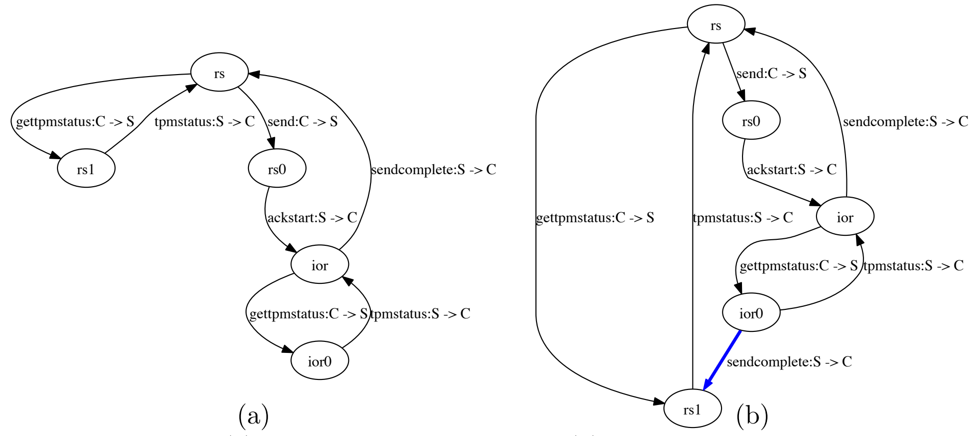


Fig. 7. (a) TpmContract Specification, (b) repaired.

5 Case Studies

We have implemented Algorithm 1 and used it to repair several un-realizable choreographies that were reported earlier [7, 17]. Our implementation obtains repaired versions of these un-realizable choreographies within a second.

Recall that the Singularity OS reservation contract (see Section 2) is un-realizable. Figure 6 presents a repaired version by adding new message exchanges. Another un-realizable contract is TpmContract (Figure 7(a)). In Figure 7(b), we show a repaired version that is automatically generated by our technique. The repaired version is similar to the one identified by authors in [9]; note however that [9] suggested an addition of a new state and two new transitions. Our repair mechanism achieves the same result by introducing one new transition between two existing states.

We have also analyzed the "Meta Conversation" protocol developed by IBM [12] and discussed in [7]. Two peers P_1 and P_2 race to decide the initiator of the interaction. The protocol is illustrated in Figure 8(a). It is un-realizable because the peers can both send the start messages (`aStartcp` and `bStartcp`)

which is not allowed in the protocol. The restriction based solution (Figure 8(b)) only allows peer P_1 to start the interaction.

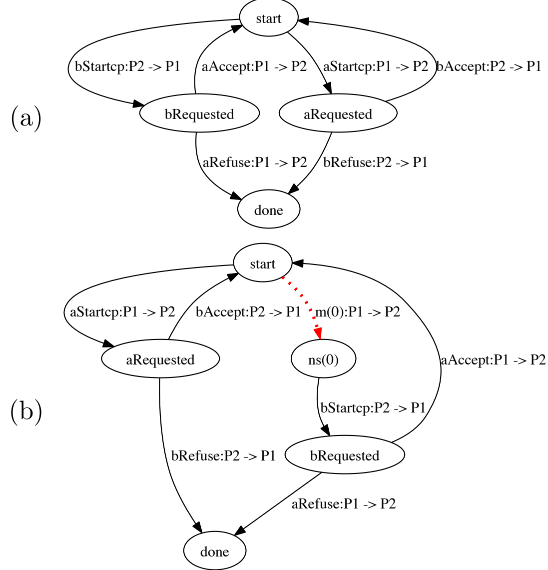


Fig. 8. (a) Meta Conversation, (b) repaired.

Note that the repair only considers the transitions and their labels, and not their semantics. For instance, in Figure 6, the added bold blue edges (relaxation) do not follow the semantics of the messages being exchanged. Consider the new path in the interaction, where "cancel" from client to server can be followed by "succeed" from the server to client. This is present in the repair in order to allow any ordering between "succeed" and "cancel" messages (as "succeed" followed by "cancel" is allowed in the original contract), which may not make sense in the context of the contract. Therefore, it is sometimes necessary to obtain certain application-domain specific information from the user such that relaxations can be guided appropriately. If the user had provided additional information that "cancel" can never be followed by "succeed", then relaxation would have been impossible and the only choice for removing difference between the un-realizable choreography and the corresponding 1-bounded system will be restriction. We allow users to provide such domain knowledge in our implementation. We have also allowed user-interaction to decide on whether relaxation or restriction is preferred for repair. The user-interaction essentially involves examination of the difference (as presented by our tool) and deciding on the choice between relaxation and restriction. Figure 9 presents an alternative solution for repairing the contract in Figure 6 generated by our tool. Observe that in this solution, a combination of relaxation and restriction has been applied.

6 Related Work

Realizability of choreographies has been studied before. The authors in [7, 9] use state machine based specifications while the authors in [10, 6] use session types; both present sufficient conditions for realizability. In [4], we have proved the decidability of choreography realizability in terms of send sequences[3] by presenting a necessary and sufficient condition for realizability.

[3] Note that, the realizability problem for the MSC-graphs, which considers both send and receive actions for realizability, is undecidable [1].

In [15], the realizability of choreography requires the developer to specify a "dominant" process for each branch and loop construct, which allows the projection mechanism to synthesize necessary synchronization messages between the dominant process and others. Similarly, techniques proposed in [14, 19, 8, 3] rely on introducing new processes, monitors and central controllers to ensure realizability. These may not be viable options if one is using a distributed computing paradigm. Moreover these techniques can be conservative in the sense that unnecessary synchronization messages can be added to even realizable choreographies. Furthermore, the focus of these works is technically different from that of our–for instance, the technique in [3] coordinates the activities of the peers in a distributed fashion such that their coordinated behavior conforms to the given choreography. The repair technique developed by authors in [13] focuses on process algebraic description of choreographies and repair by restriction in the context of independent sequences (referred to as connected choreography by the authors); additionally, the description does not take into consideration iterations, which makes the technique inapplicable to choreographies with cycles.

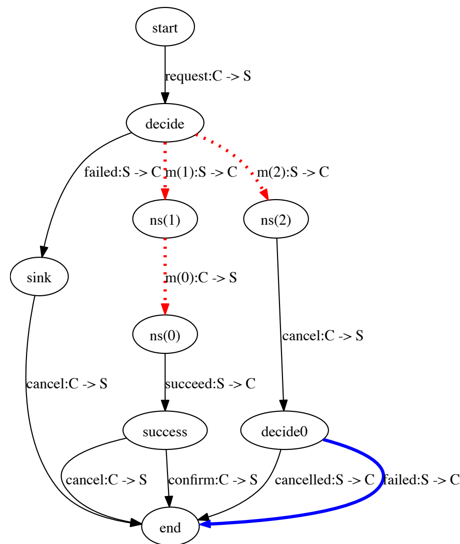


Fig. 9. Alternative repair strategy for ReservationSession (Figure 2(b))

In contrast, our work (which includes both relaxation and restriction mechanisms) does not require introduction of new processes, does not require a central controller, and does not require use of synchronous communication between any entities/peers. As our technique is based on finite state machines and their language equivalence, it is applicable to choreographies and interactions which are specified at different levels of abstractions, such as session-types [10] and collaboration diagrams [5], as long as these specifications are translated to state-machine based representation described in [4] and used in this paper.

7 Conclusion

We present techniques for automatically repairing un-realizable choreographies based on two strategies: 1) relaxation, where new behaviors are added to the choreography as part of the repair and 2) restriction, where un-desired (excluded by the choreography) behaviors in the system obtained by projecting the choreography are removed as part of the repair. We prove that our repair algorithm always terminates with a realizable choreography. To the best of our knowledge, our method is the first to consider automatically repairing choreographies and to provide formal guarantees of correctness.

References

1. R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of MSC graphs. In *Proc. 28th Int. Colloq. on Automata, Languages, and Programming*, pages 797–808, 2001.
2. J. Armstrong. Getting Erlang to talk to the outside world. In *Proc. ACM SIGPLAN Workshop on Erlang*, pages 64–72, 2002.
3. M. Autili, D. Ruscio, A. Salle, P. Inverardi, and M. Tivoli. A model-based synthesis process for choreography realizability enforcement. In *Fundamental Approaches to Software Engineering*, pages 37–52, 2013.
4. S. Basu, T. Bultan, and M. Ouederni. Deciding choreography realizability. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2012.
5. T. Bultan and X. Fu. Specification of realizable service conversations using collaboration diagrams. In *Service Oriented Computing and Applications*, 2008.
6. P.-M. Denielou and N. Yoshida. Multiparty session types meet communicating automata. In *In Proceedings of ESOP*, 2012.
7. X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specification and verification of reactive electronic services. In *Proc. of the 8th Int. Conf. on Implementation and Application of Automata (CIAA)*, 2003.
8. M. Güdemann, G. Salaün, and M. Ouederni. Counterexample guided synthesis of monitors for realizability enforcement. In *Automated Technology for Verification and Analysis*, pages 238–253. Springer, 2012.
9. S. Hallé and T. Bultan. Realizability analysis for message-based interactions using shared-state projections. In *SIGSOFT Foundations of Software Engineering*, 2010.
10. K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *Proceedings of Symposium Principles of Programming Languages*, 2008.
11. G. C. Hunt and J. R. Larus. Singularity: rethinking the software stack. *Operating Systems Review*, 41(2):37–49, 2007.
12. S. Kumaran, P. Nandi, J. Hanson, T. Heath, and Y. Patnaik. Conversational browser. IBM Techreport, 2004.
13. I. Lanese, F. Montesi, and G. Zavattaro. Amending choreographies. In *Automated Specification and Verification of Web Systems*, 2013.
14. N. Lohmann and K. Wolf. Realizability is Controllability. In *Proc. 1st Central-European Work. on Services and Their Composition*, pages 61–67, 2009.
15. Z. Qiu, X. Zhao, C. Cai, and H. Yang. Towards the theoretical foundation of choreography. In *In Proceedings of Conference on World Wide Web*, 2007.
16. Singularity design note 5 : Channel contracts. singularity rdk documentation (v1.1). `http://www.codeplex.com/singularity`, 2004.
17. Z. Stengel and T. Bultan. Analyzing singularity channel contracts. In *Proc. 18th Int. Symp. on Software Testing and Analysis (ISSTA)*, pages 13–24, 2009.
18. Web Service Choreography Description Language (WS-CDL). http://www.w3.org/TR/ws-cdl-10/, 2005.
19. Y. Yoon, C. Ye, and H.-A. Jacobsen. A distributed framework for reliable and efficient service choreographies. In *Proceedings of the 20th International Conference on World wide web*, WWW '11, pages 785–794. ACM, 2011.