Non-desynchronizable Service Choreographies

Gero Decker[1], Alistair Barros[2], Frank Michael Kraft[3], and Niels Lohmann[4]

[1] Hasso-Plattner-Institute, University of Potsdam, Germany
gero.decker@hpi.uni-potsdam.de
[2] SAP Research Centre, Brisbane, Australia
alistair.barros@sap.com
[3] SAP AG, Walldorf, Germany
frank.michael.kraft@sap.com
[4] Institut für Informatik, Universität Rostock, 18051 Rostock, Germany
niels.lohmann@uni-rostock.de

Abstract. A precise definition of interaction behavior between services is a prerequisite for successful business-to-business integration. Service choreographies provide a view on message exchanges and their ordering constraints from a global perspective. Assuming message sending and receiving as one atomic step allows to reduce the modelers' effort. As downside, problematic race conditions resulting in deadlocks might appear when realizing the choreography using services that exchange messages asynchronously. This paper presents typical issues when desynchronizing service choreographies. Solutions from practice are discussed and a formal approach based on Petri nets is introduced for identifying desynchronizable choreographies.

1 Introduction

The service oriented architecture (SOA) is an architectural style for building software systems based on services. Services are loosely coupled components described in a uniform way that can be discovered and composed. One realization of a SOA is the web services platform architecture where services are offered as web services [1].

In a first generation of services only pairs of request/response message exchanges were considered. This view is sufficient when considering simple services, for instance, a stock information service, where the current or a past value of a share can be requested. However, more complex interactions must be considered in many real-world business scenarios. For instance, in a typical purchasing scenario, goods can be ordered, orders can be modified or canceled, orders must be acknowledged and delivery can be rerouted, or alternative products or quantities are offered in out-of-stock situations. Also multi-lateral scenarios involving, for instance, external payment, shipment and insurance services need to be considered. These scenarios involve multiple interactions and the complex dependencies between them must be addressed.

A. Bouguettaya, I. Krueger, and T. Margaria (Eds.): ICSOC 2008, LNCS 5364, pp. 331–346, 2008.

Service choreographies are a means to specify the messages exchanged between different services and the dependencies between them. Even multi-lateral scenarios can be captured with typical languages such as the Web Service Choreography Description Language (WS-CDL [2]). Here, interactions between services are the atomic building blocks and ordering constraints are defined between them. That is, sending and receiving of messages are modeled as one step. The ordering constraints then define what other interactions must have occurred before a certain interaction. In the remainder we will refer to this modeling style as *interaction modeling*.

Interaction modeling, as opposed to distinguishing message sending and reception as separate activities in the control flow, results in several advantages for the modeler. (i) Control flow dependencies do not need to be specified per service, but rather as seen from the perspective of an ideal observer. That way redundant control flow relationships are avoided and the chance of modeling deadlock situations is minimized. Furthermore, avoiding redundant structures allows for faster model creation. (ii) Branching structures can be specified globally, that way avoiding modeling errors caused by incompatible branching structures [3].

As a downside of interaction modeling, the intuitive interpretation that all interactions are atomic steps hides certain challenges that arise when taking the choreography to an asynchronous world. Especially in situations where there exists a mutual exclusion between two interactions with different senders, i. e. *mixed choices*, the asynchronous nature of message exchanges might cause severe problems due to race conditions.

This paper addresses the issue of race problems in asynchronous settings. It discusses typical solutions in real-world applications and presents a formal framework for detecting and locating race conditions in service choreographies.

The remainder of this paper is structured as follows. The next section will present a real-world example where mixed choices actually lead to deadlocks. Section 4 lists and discusses typical solutions in practice, before Sect. 3 introduces a formal framework for identifying and locating problematic race conditions. Section 5 reports on related work in this area and Sect. 6 concludes.

2 Motivating Example

A purchase order process looks as follows. A buyer submits an order to a seller. The seller returns a confirmation message to the buyer. Finally, delivery and payment are carried out. The buyer and the seller are both realized as services.

Several situations require a deviation from this simple process. While the seller has not sent a delivery notification and the buyer has not initiated payment yet, the buyer has the possibility to issue a change request, e. g. demanding an increased quantity. A change request must be acknowledged by the seller. The buyer might as well cancel the order which in turn needs to be confirmed by the seller. On the other hand, there might be a seller initiated change proposal, e. g. the previous confirmation is revised by proposing a delay of the delivery date.

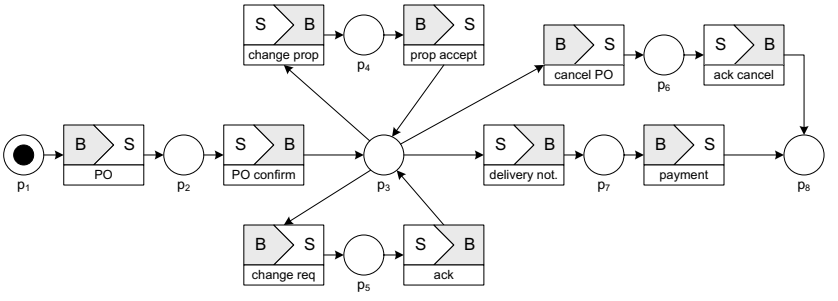


Fig. 1. Purchase order choreography

Figure 1 illustrates these interactions using the interaction Petri net notation from [4]. Each rectangle represents the message exchange between a sender (upper left corner) and a receiver (upper right corner). The label at the bottom of a rectangle indicates the message type. The circles represent places that can contain tokens. The token flow defines the control ordering constraints between message exchanges belonging to the same choreography instance.

The interaction Petri net assumes a world where message send and receive happen in one atomic step. This assumption is not valid in many scenarios. In our example, the different services might send messages concurrently. For example, the buyer's and seller's decisions to send change requests or change proposals are decoupled and therefore it is a common scenario that the services send messages concurrently. Therefore, we need to desynchronize the choreography to properly reflect that message sending and receiving are separate steps. Figure 2 shows a corresponding Petri net.

The original purchase order model does not contain any obvious problems. Each choreography instance eventually terminates in a state where there is one token left on place p_8 of the net. The desynchronized model, on the other hand,

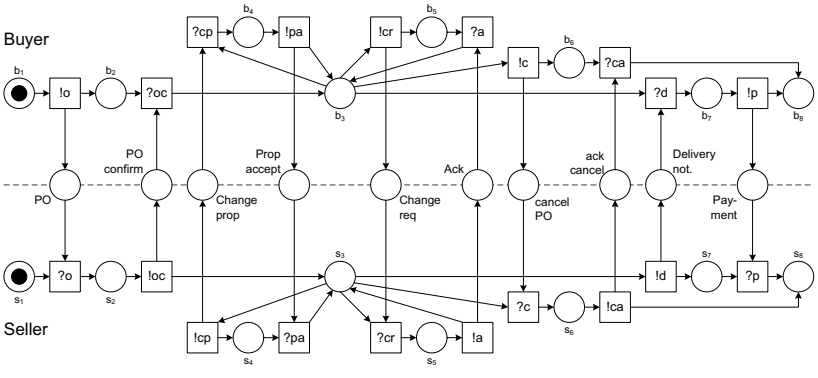


Fig. 2. Desynchronized purchase order choreography

contains several problems. For instance, a deadlock occurs if the buyer decides to cancel the order and the seller proposes a change. Here, the buyer would wait infinitely for a confirmation of the cancellation and the seller waits infinitely for the response to his proposal.

The issues presented in this example are not specific to purchase ordering scenarios. Similar issues can be found in many other areas of enterprise systems. In order to detect and locate such race problems, we introduce a formal approach in the next section.

3 Formal Model

This section will introduce a formal framework for detecting and locating problematic race conditions in service choreographies. The framework builds upon Interaction Petri nets (IPNs), a special kind of labeled Petri nets, where a transition models the message exchange between two services. IPNs have been proposed in [4]. While the example presented in Sect. 2 is bilateral, IPNs can also be used to represent multi-lateral choreographies, i. e. more than two roles are involved in the choreography. IPNs have the same token flow semantics as classical Petri nets [5] and concentrate on the control and message flow aspect of choreographies.

Definition 1 (Petri Net). *A* Petri net *is a tuple* $[P, T, F, m_0]$ *where* P *and* T *are two finite disjoint sets of places and transitions, respectively,* $F \subseteq \big((P \times T) \cup (T \times P)\big)$ *is a flow relation, and* $m_0 : P \to \mathbb{N}$ *is an initial marking.*

We assume the standard firing rule of Petri nets, and write $m \xrightarrow{t} m'$ if the marking m' is reachable from marking m by firing transition t. Reachability can be canonically extended to transition sequences.

Message types are first-class constructs in IPNs, allowing the distinction between, for example, acceptance and rejection messages. In the following definitions, we denote the set of all roles by R. A concrete service participating in a choreography instance plays one or several roles, for example, "buyer" or "seller". The set of message types is denoted by MT.

Definition 2 (Interaction Petri Net). *An* Interaction Petri net (IPN) *is a tuple* $N = [P, T, F, m_0, final, \lambda]$ *where*

- $[P, T, F, m_0]$ *is a Petri net,*
- *final is a finite set of valid final markings, and*
- $\lambda : T \to (R \times R \times MT) \cup \{\tau\}$ *is a labeling function assigning a sender role, a receiver role and a message type to a transition, or declaring it as silent transition.*

The IPN in Fig. 1, has two roles $R = \{\mathsf{B}, \mathsf{S}\}$, ten message types $MT = \{\mathsf{PO}, \mathsf{PO\ confirm}, \ldots, \mathsf{payment}\}$, and $[\mathsf{p}_8]$ (i. e., one token on place p_8) is the only final marking. With the help of final markings, we can differentiate desired final states from unwanted deadlocks. This can be expressed with the concept of *weak termination*.

Definition 3 (Weak Termination). *An interaction Petri net* weakly terminates *iff, from every marking reachable from m_0, a final marking $m_f \in final$ is reachable.*

It can be easily checked that the IPN in Fig. 1 weakly terminates; that is, the marking $[\mathsf{p}_8]$ can be reached from all reachable markings.

The following definition bridges service choreographies and the local views on such a choreography, i. e. the subset of interactions and control flow dependencies that are relevant for a given role.

Definition 4 (Role Projection). *Let N_1 and N_2 be interaction Petri nets. N_1 is the* projection of N_2 for role r *iff $\forall t \in T_1 : \lambda(t) = \tau \vee r \in \lambda(t)$, and there exists a relation Q between the markings of N_1 and N_2 such that:*

i. $m_{01}\ Q\ m_{02}$,
ii. *if* $m_1\ Q\ m_2$, $m_1 \xrightarrow{t}_{N_1} m_1'$ *and* $r \notin \lambda(t)$, *then* $m_1'\ Q\ m_2$,
iii. *if* $m_1\ Q\ m_2$, $m_1 \xrightarrow{t}_{N_1} m_1'$ *and* $r \in \lambda(t)$, *then* $m_2 \overset{t}{\Longrightarrow}_{N_2} m_2'$ *and* $m_1'\ Q\ m_2'$,
iv. *if* $m_1\ Q\ m_2$, $m_2 \overset{t}{\Longrightarrow}_{N_2} m_2'$ *and* $r \in \lambda(t)$, *then* $m_1 \overset{t}{\Longrightarrow}_{N_1} m_1'$ *and* $m_1'\ Q\ m_2'$,
v. *if* $m_1\ Q\ m_2$ *and* $m_1 \in final_1$, *then* $m_2 \Longrightarrow_{N_2} m_2'$ *and* $m_2' \in final_2$,
vi. *if* $m_1\ Q\ m_2$ *and* $m_2 \in final_2$, *then* $m_1 \Longrightarrow_{N_1} m_1'$ *and* $m_1' \in final_1$,

where $m \Longrightarrow m'$ denotes that there exists a (potentially empty) firing sequence of transitions $t_1, \ldots, t_n$ from m to m' where for all t_i holds $r \notin \lambda(t_i)$ and $m \overset{t}{\Longrightarrow} m'$ denotes $m \overset{t}{\Longrightarrow} m'$. Furthermore, $r \in \lambda(t)$ denotes that r is the sending or receiving role of t.

Role projection ensures that the order of communication actions possible in the local view correspond to the order of interactions as specified in the choreography. By considering the branching structures, role projection goes beyond trace comparison. The relation is similar to branching bisimulation [6], while there are key differences: Whenever branching within a service depends on a choice done by other services, no internal decision must be made. On top of that, role projection relates the sets of final markings.

The projection algorithm in [4] preserves role projection. This algorithm projects a choreography by applying transformation rules on the structure of the interaction Petri net.

We assume that a service choreography is *realizable* [7,4]. This ensures that there is indeed a set of local views that, assuming synchronous communication, collectively show exactly the behavior as specified in the choreography.

An IPN N_s can be "desynchronized" to a net N_a using the role projections of N_s. Thereby, we introduce a place p_α for each message type α of the IPN N_s which models an asynchronous message channel and is connected to the sender and receiver according to the roles. Whereas communication is atomic in N_s, the sending and receipt of a message is x explicitly modeled by two transitions $!x$ and $?x$ of N_a.

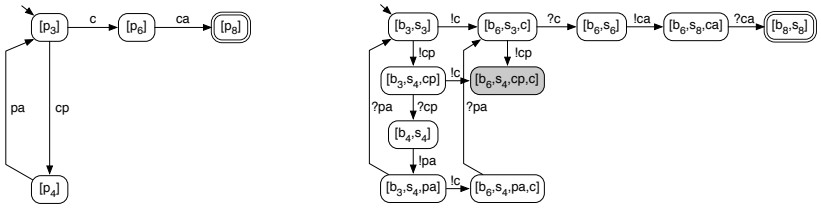
Definition 5 (Desynchronized Net). *Let N_s be an IPN and $R = \{r_1, \ldots, r_n\}$ the set of all roles involved in N_s. The IPN N_a is a* desynchronized net *for N_s iff, for all $i = 1, \ldots, n$, N_i are pairwise disjoint role projections of N_s for roles r_i, and*

- $P_a = \bigcup_i P_i \cup \{p_\alpha \mid \exists t \in T_s : (\lambda(t) \neq \tau \wedge \alpha = \lambda(t))\}$,
- $T_a = \bigcup_i T_i$,
- $F_a = \bigcup_i F_i \cup \{(t, p_\alpha) \mid \exists x, mt\ (\lambda(t) = (r_i, x, mt))\} \cup \{(p_\alpha, t) \mid \exists x, mt\ (\lambda(t) = (x, r_i, mt))\}$,
- $m_{0a} = m_{01} \oplus \cdots \oplus m_{0n}$,
- $final_a = \{m_{f1} \oplus \cdots \oplus m_{fn} \mid m_{fi} \in final_i\}$, *and*
- $\lambda_a(t) = \lambda_i(t)$ *iff* $t \in T_i$.

The composition of markings is defined as $m_1 \oplus \cdots \oplus m_n(p) = m_i(p)$ *iff* $p \in P_i$.

The desynchronized net N_a usually has more behavior than the original IPN N_s: The atomic message transfer in N_s can be mimicked by N_a by firing first the sending and then the receiving transition. Moreover, it might also be possible that N_a can fire a transition in an intermediate state introduced by the decoupling of sender and receiver. If this additional behavior does not jeopardize weak termination, N_s is *desynchronizable*.

Definition 6 (Desynchronizability). *Let N_s be a weakly terminating IPN. N_s is* desynchronizable *iff there exists a desynchronized net N_a for N_s that weakly terminates.*



(a) synchronous choreography (b) asynchronous choreography with deadlock

Fig. 3. Reachability graphs showing the example's race problem

The most frequent reason for non-desynchronizable choreographies are mixed choices, where there is a conflict between transitions with different senders. As mentioned earlier, the desynchronized example choreography (see Fig. 2) contains a deadlock. Thus, the original choreography in Fig. 1 is not desynchronizable. This can be detected by analyzing the reachability graphs of the nets. In Fig. 3(a), a part of the reachability graph of the original choreography is depicted. In the marking $[\mathsf{p}_3]$ the transitions c and cp are (among others) enabled. The same situation is depicted in Fig. 3(b) for the desynchronized net. Here, the transitions $!\mathsf{c}$ and $!\mathsf{cp}$ are enabled in $[\mathsf{b}_3, \mathsf{s}_3]$, but can occur concurrently: neither transition disables the other, and a deadlocking marking $[\mathsf{b}_6, \mathsf{s}_4, \mathsf{cp}, \mathsf{c}]$ is reachable when firing these transitions in either order.

Definition 7 (Conflict Transitions). *Let N_s be a non-desynchronizable interaction Petri net and N_a a desynchronized net for N_s. Define the set of* conflict transitions *T_C to contain all transitions t of N_a such that:*

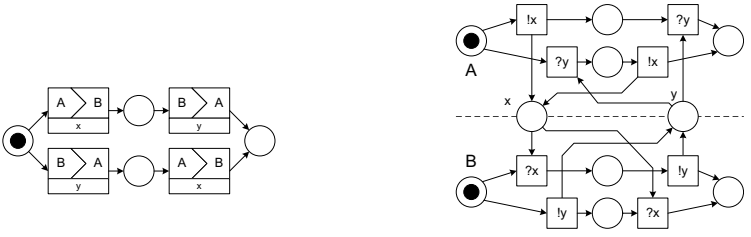
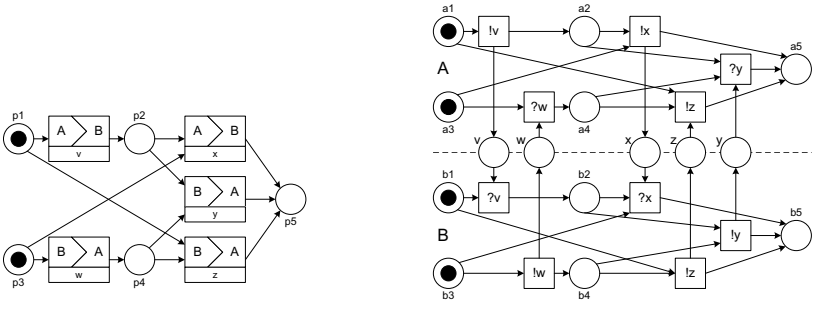


Fig. 4. Structural conflicts do not necessarily lead to deadlocks

– *there exists a marking* m *with* $m \xrightarrow{*} m_f$ *for a marking* $m_f \in final$*, and*
– *there exists a marking* m' *with* $m \xrightarrow{t} m'$ *and* $m' \not\xrightarrow{*} m'_f$ *for any* $m'_f \in final$*.*

The set T_C contains all transitions whose firings can make a final marking unreachable. From Fig. 3(b) we can conclude that the transitions !c and !cp are conflict transitions for the desynchronized net of Fig. 2. With state-of-the-art Petri net model checkers such as LoLA [8], race problems can be detected efficiently even for larger choreographies.



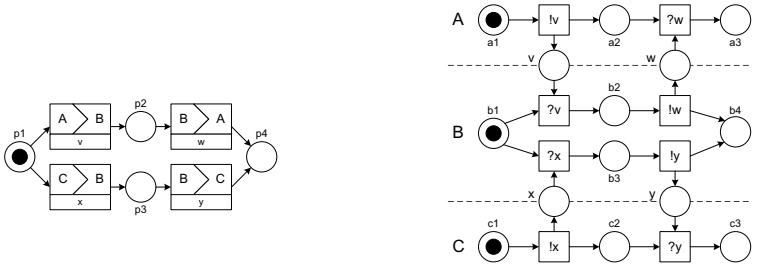
(a) synchronous choreography (b) desynchronized choreography

Fig. 5. Conflict transitions without structural conflict

While mixed choices are a typical reason for race problems, not all mixed choices are problematic (see Fig. 4). Here, both services can send a message before receiving one. However, due to the follow-up interactions, the desynchronized choreography weakly terminates.

Finally, conflict transitions do not necessarily need to be in a structural conflict, i. e. sharing common input places. Figure 5(a) shows a synchronous choreography that weakly terminates with final marking [p5]. Here, the choice whether the transition regarding message v or the transition regarding w fires first influences what transitions will be enabled later on. If the transition involving v fires, the transition involving z will not be enabled any longer.

The definition of conflict transitions also captures those scenarios where individual services are able to send messages in a final marking. Figure 6(a) shows



(a) synchronous choreography (b) desynchronized choreography

Fig. 6. Deadlocks caused by firing transitions in a valid final marking

an example involving three roles and final marking [p4]. Figure 6(b) shows the desynchronized choreography as generated by the algorithm in [4]. The final markings for the individual role projections are $[a_1]$ and $[a_3]$ for A, $[b_4]$ for B and $[c_1]$ and $[c_3]$ for C. This results in the valid final markings $[a_1, b_4, c_3]$, $[a_3, b_4, c_1]$, $[a_1, b_4, c_1]$, and $[a_3, b_4, c_3]$ for the desynchronized choreography, while only the first two markings are actually reachable. In marking $[a_1, b_4, c_3]$ role A is ready to fire transition !v. Firing this transition actually leads to a marking from where no valid final marking can be reached any more. Therefore, !v is a conflict transition.

4 Typical Resolutions to Race Problems

The definitions from the previous section enable us to locate conflict transitions. As a next step, one or several strategies have to be chosen to remove race problems from a choreography. Instead of formally proposing one particular strategy or defining an algorithm to automatically choose among different possible strategies, we rather sketch several strategies applied in real-world implementations in this section. Each strategy comes with a set of implications on the business level that need to be carefully considered before applying them.

In the remainder of this section we will use the term *conflicting messages* for a pair of messages sent by different partners that correspond to a pair of conflict transitions as defined in the previous section. Both messages must belong to the same choreography instance.

It could be a possible strategy to resolve the choreography in such a way that conflicting messages simply cannot occur any longer. For the example, this would mean that there is no chance of having a delivery notification and a cancel message been sent in the same choreography instance. This could be achieved for instance through total sequentialization of the choreography, where only one partner is allowed to send messages at a time. However, such a strategy is typically not feasible in real-world scenarios. It is often desired that conflicting messages are possible but for the case that this occurs, a predefined resolution must be in place. Therefore, we are going to list different strategies applied in practice that follow this approach.

The following strategies can be categorized into two groups. Either (a) there is a predefined outcome upon conflicting messages, most typically one message is considered and the others are ignored, or there might be different outcomes possible. Here, we can again distinguish three types: (b) one partner could be allowed to determine the outcome and tell the other partners his decision; it could also be defined that (c) each partner decides individually for the outcome, or that (d) there is a negotiation regarding the desired outcome.

4.1 Precedence

The general idea is to define precedence relationships at design-time, prescribing how partners have to behave in the case of conflicting messages. Therefore, precedence mostly falls into category (a). If a partner detects conflict messages, he knows the outcome of this conflict and can immediately continue accordingly. He assumes that the other partners will also detect this conflict sooner or later and also act accordingly.

The definition of precedence relationships must not be seen as pure technicality as it directly has business impact. Therefore, precedence relationships would need to be part of interaction contracts. Regarding the definition of precedence relationships we distinguish three different strategies.

Singular Interaction Partner Precedence. This strategy looks at individual interactions, e. g. the cancellation interaction in our example, and defines precedence of one partner over the other. Here, we can distinguish between two settings: (i) the buyer has precedence over the seller or (ii) the seller has precedence over the buyer.

Case (i) means that if the buyer sends the cancellation, the seller has to accept the buyer's cancellation in any case. This means that the buyer can assume that the cancellation message will have the desired effect, once it has been sent. Therefore, the seller does not need to return any confirmation message in this case. This corresponds to category (a).

In case (ii) the seller has a veto right regarding cancellation messages sent by the buyer. The seller can accept this request and return a cancellation confirmation. Only now the buyer can be sure that cancellation was successful. As an alternative, the seller could also send a cancellation rejection. Therefore, the seller can decide on the outcome, implying category (b).

Deciding for each interaction for a partner precedence individually does not solve race problems in the general case. If, for example, we decide that the buyer has precedence regarding buyer initiated cancellation and the seller has precedence regarding seller initiated change proposals, deadlocks are still possible. Now imagine the opposite setting where the seller has precedence regarding buyer initiated cancellation and the buyer has precedence regarding seller initiated change proposals. Here, the partners have veto rights for the corresponding requests. If the buyer sends a cancellation request and the seller sends a change proposal at the same time, the buyer will reject the change proposal as it conflicts with the previously sent cancellation request. The same holds true for the

seller reacting to the cancellation request. After both partners have rejected the respective requests, they can, of course, resend their requests.

Type-Based Precedence between Multiple Interactions. While the previous strategy considered interactions individually, this strategy considers precedence regarding combinations of interactions. Here, the message types are considered and always a fixed outcome is defined, therefore category (a). A crucial aspect of this strategy is that no combination of interactions is forgotten.

A precedence rule could be that a delivery notification has precedence over buyer initiated cancellation messages and buyer initiated change requests. On the other hand, a buyer initiated request always has precedence over seller initiated change proposals. Figure 7 illustrates a resolved desynchronized choreography for this precedence rule. This resolved version weakly terminates. All transitions that were added to the original Petri net with striped background.

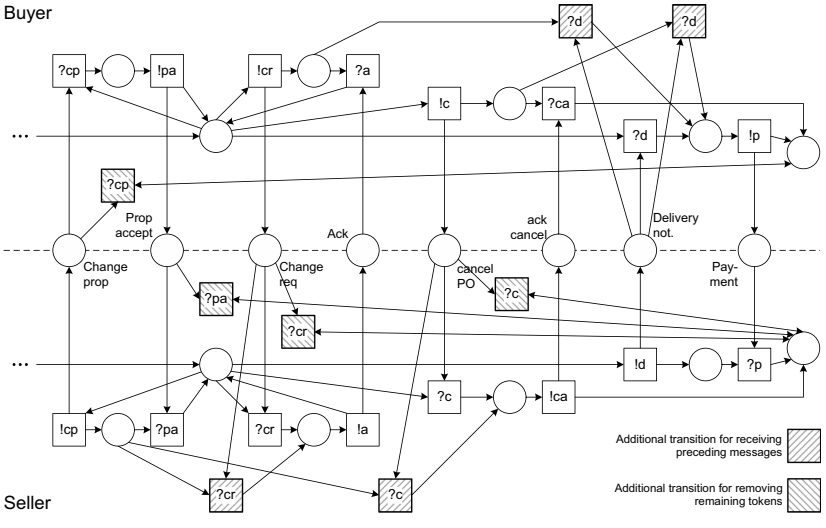


Fig. 7. Resolved purchase order choreography

A conflict between a seller initiated change proposal and a buyer initiated change request is resolved in the following way. In addition to being ready to consume an acceptance message for the change proposal, the seller can also consume a change request or a cancel message instead. This is manifested through the additional transitions ?cr and ?c transitions. The change proposal message must finally be consumed by the buyer without having any effect on the buyer. This happens through the additional ?cp transition.

The proposed solution in Fig. 7 is still not optimal from a business point of view. If the seller sends a change proposal while the buyer sends a change request, the messages conflict and the seller will receive the change request and accept

it. In this situation, the seller assumes that the buyer will ignore the change proposal. However, the buyer could receive the accept message first and receive the change proposal afterwards. Now, the buyer cannot know that this change proposal conflicted with the change request and therefore accepts the proposal. However, the seller is not able to receive this message and will only remove the remaining token at the end of the choreography. From a business point of view this behavior is undesired: the buyer has accepted a change proposal that the seller assumes obsolete.

Another problem might arise when precedences are cyclic: Imagine there are three partners A, B and C. A can send a message to B (interaction *ab*), B to C (*bc*), and C to A (*ca*). *bc* has precedence over *ab*, *ca* has precedence over *bc*, and *ab* has precedence over *ca*. Now a conflict involving all three interactions occurs. Every partner thinks that his message has precedence over the message received and simply ignores the incoming message. This again could result in a deadlock.

Situation-Based Precedence between Multiple Interactions. While precedence rules between different interactions were based on message types, this strategy allows more fine-grained precedence rules and again falls into category (a).

Imagine a logistics scenario where a customer lets a shipper transport his goods. The shipper selects different carriers and creates a shipment that he sends to the customer and which needs to be commented by the customer. At the same time, the customer has the possibility to cancel his order. While cancellation precedes the shipment plan interaction in the default case, this is only true for the first two weeks after the initial order. After these two weeks have passed, the shipment plan interaction precedes the cancellation. This might be due to the fact that cancellation at this point in time would simply involve too much cost. However, while the shipment plan has not been finalized yet, the customer can still cancel the order.

An underlying assumption of this strategy is that both partners come to the same conclusion about precedence. As time is the criterion in this example, both need common understanding about when the two weeks have passed. Therefore, the arrival time of the message cannot be used as criterion, as the corresponding sender might not be able to know when this is.

4.2 Allowing Individual Decisions

Allowing individual decisions leaves it open to every partner involved to decide for an outcome individually: category (c). In the case of a buyer initiated cancellation request conflicting with a seller initiated change request, there are two alternatives for each partner:

- The seller either (S1) rejects the cancellation request and assumes that the change request has still relevance or (S2) accepts the cancellation request and assumes that the change request is obsolete.

– The buyer either (B1) rejects the change request and assumes that the cancellation request has still relevance or (B2) accepts the change request and assumes that the cancellation request is obsolete.

Out of these possibilities two are ideal outcomes: The combinations (S1)+(B2) and (S2)+(B1) lead to the acceptance of exactly one request. Even the combination (S1)+(B1) is acceptable, as the choreography instance is exactly in the same state as before the two requests and requests can be issued again. Maybe this time, one of the partner succeeds with his intent.

Only the combination (S2)+(B2) is problematic as both requests were accepted and both partners assume a wrong situation. However, once an accept message finally arrives, the conflict is detected and a resolution can be achieved as described in the other strategies.

Although this strategy does not guarantee a proper resolution in the general case and requires resorting to other resolution strategies, it is still worth considering as most outcomes are acceptable. A major challenge of this approach is that the process instances need to be realigned in case a partner has already continued, assuming his decision led to an acceptable situation. This might involve compensation and becomes especially difficult if communication to other partners is involved.

4.3 Negotiation of Outcome

Negotiation is another strategy where the outcome of conflicting messages is not fixed, therefore category (d). Here, the different partners need to reach agreement about the outcome. Such negotiation can either happen through human intervention or automatically. Human intervention could simply involve a phone call or an email exchange. In many cases such human intervention is actually desirable. For instance, the cost of cancellation might increase depending on what actions the partner has already performed. Therefore, it could be negotiated whether cancellation is still desired under the new conditions.

As an alternative, a formal hand-shake to support negotiation could be factored into each partner's process. For this, all partners need to agree on conflicting messages requiring negotiation and implement common exception handling logic. This would involve strictly sequential interactions, as partners arbitrarily reciprocate to resolve the conflict. First, conflicting messages would be detected by a partner and be broadcast to relevant partners. Each partners process would be required to escalate to its common exception handling logic such that all parts of the process impacted by the conflict are suspended. The first part of the exception handling logic would be to determine which partner gets the write token. This remains an open issue although some basic heuristics could be defined, e. g. the first partner detecting the conflicting messages gets the write token. Another serious issue is managing multiple conflicts which can arise concurrently and determining the priority in which they should be handled. These and other issues have been handled in techniques applied for self-stabilizing systems [9].

5 Related Work

Different service choreography languages have been proposed that follow the interaction modeling style. The Web Service Choreography Description Language (WS-CDL [2]) is a standard proposed by the World Wide Web Consortium. Alternative proposals from academia are Let's Dance [10] and the Interactive Systems Description Language (ISDL [11]). The issue of race problems when taking choreographies defined in these languages to an asynchronous world has not been tackled so far.

There are different formal models available for describing choreographies. A survey can be found in [12], where a distinction between automata-based, Petri-net-based and process-algebra-based approaches is made. Most approaches include techniques for relating choreographies to models describing the behavior of individual services. For instance, [13] use a bisimulation-like relation to check conformance between a local model and the choreography.

There has been extensive research in the area of compatibility checking, where the absence of deadlocks is of central importance, e. g. [14,15,16,17]. While detecting and locating deadlocks is covered by most approaches, more novel approaches deal with the question of automatically repairing faulty choreographies [18]. While such approaches could indeed be used to repair the choreography presented in Sect. 2, any outcome would include forbidding certain messages. Either the buyer must not send a change request or a cancellation message, or the seller must not send a change proposal or a delivery notification. Such a solution is primarily aimed to explain faulty choreographies by proposing fixed versions, but would, of course, be inacceptable from a business point of view.

The issue of desynchronizability is closely related to the question of synchronizability of asynchronous choreographies [19]. If an asynchronous choreography is synchronizable then the same set of choreography instances are produced under asynchronous and synchronous communication semantics. If applied in a top-down manner, i. e. a synchronous choreography is projected to an asynchronous choreography, synchronizability can be used to detect race problems as presented in this paper. However, our approach goes beyond synchronizability analysis as it allows to locate the reasons of desynchronizability. This is important as there might be different sets of conflict transitions that might be treated in isolation of each other, which in turn might allow for a less-invasive resolution.

The problem of mixed choices between sending and receiving has been studied in the context of distributed message protocols and algorithms [20]. For example, the *crosstalk algorithm* adds round numbers to each sent message which help to identify and solve conflicts. Again, such protocols do not take the content and the original choreography into account and thus are not suitable to solve the problem from a business point of view.

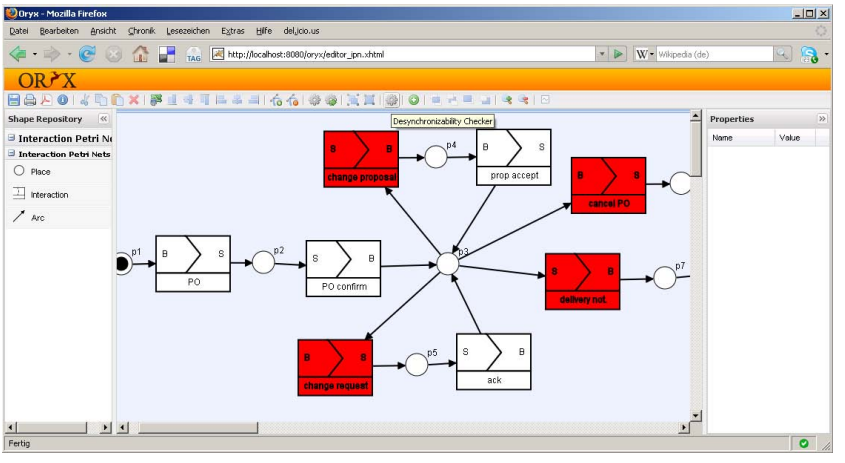


Fig. 8. Screenshot of the interaction Petri net modeler and desynchronizability checker

6 Conclusion

This paper has discussed the issue of non-desynchronizability in choreographies where message send and receive activities are considered as atomic steps. We introduced a formal approach for detecting and locating potentially conflicting message exchanges. This approach is based on interaction Petri nets.

We have implemented a tool that realizes this approach. As an extension to the web-based modeling platform Oryx[1], interaction Petri net stencils and a plug-in for desynchronizability checking were developed. Figure 8 shows a screenshot of the tool, where the conflict transitions of the initial example are highlighted in red and bold labels. In future work, we will extend the tool chain to desynchronizability analysis for iBPMN choreographies. iBPMN is an extension for the Business Process Modeling Notation (BPMN [21]), allowing for interaction modeling in a BPMN-like notation [3].

The resolution of race problems is not a pure technical issue that can be carried out late in the actual development of services. The discussion of typical resolution strategies in Sect. 4 has shown business implications when choosing one strategy or another. Therefore, precedence relationships, for instance, would have to be discussed and defined in very early choreography design stages.

The current solution to non-desynchronizable choreographies looks as follows: An interaction model is created, then desynchronizability is verified as presented in Sect. 3. If desynchronizability is detected the desynchronized choreography can directly be used as starting point for implementing services or adapting existing ones. If this is not the case, manual resolution along one of the resolution strategies has to be carried out for the desynchronized choreography first.

[1] See `http://oryx-editor.org`.

One might argue that we do not need interaction models all together as we have to resort to asynchronous models anyway (in the case of non-desynchronizability), and we should rather use asynchronous models from the beginning. However, the advantages gained for the desynchronizable parts of the choreography are already immense. Having the possibility to generate the desynchronized model increases modeling speed.

A classification of different resolution strategies shows the vision for dealing with non-desynchronizability. Ideally, modelers can choose from a predefined set of strategies, being informed about the business implications of a chosen strategy. Such a declarative approach would finally result in generating fully resolved choreographies. With such an approach the modelers would not need to touch the generated model any longer. Therefore, future work will center around formally refining the different strategies and proposing a declarative framework for the resolution of race problems.

References

1. Curbera, F., Leymann, F., Storey, T., Ferguson, D., Weerawarana, S.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall PTR, Englewood Cliffs (2005)
2. Kavantzas, N., Burdett, D., Ritzinger, G., Lafon, Y.: Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation. Technical report (2005), `http://www.w3.org/TR/ws-cdl-10`
3. Decker, G., Barros, A.: Interaction Modeling using BPMN. In: ter Hofstede, A.H.M., Benatallah, B., Paik, H.-Y. (eds.) BPM Workshops 2007. LNCS, vol. 4928, pp. 208–219. Springer, Heidelberg (2008)
4. Decker, G., Weske, M.: Local enforceability in Interaction Petri Nets. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 305–319. Springer, Heidelberg (2007)
5. Reisig, W.: Petri Nets. Springer, Heidelberg (1985)
6. van Glabbeek, R.J., Weijland, W.P.: Branching time and abstraction in bisimulation semantics. J. ACM 43(3), 555–600 (1996)
7. Fu, X., Bultan, T., Su, J.: Conversation protocols: A formalism for specification and analysis of reactive electronic services. Theor. Comput. Sci. 328(1-2), 19–37 (2004)
8. Schmidt, K.: LoLA: A Low Level Analyser. In: Nielsen, M., Simpson, D. (eds.) ICATPN 2000. LNCS, vol. 1825, pp. 465–474. Springer, Heidelberg (2000)
9. Dolev, S.: Self-stabilization. MIT Press, Cambridge (2000)
10. Zaha, J.M., Barros, A., Dumas, M., ter Hofstede, A.H.M.: Let's Dance: A language for service behavior modeling. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 145–162. Springer, Heidelberg (2006)
11. Quartel, D., Dijkman, R., van Sinderen, M.: Methodological support for service-oriented design with ISDL. In: Proc. ICSOC 2004, pp. 1–10. ACM, New York (2004)
12. Su, J., Bultan, T., Fu, X., Zhao, X.: Towards a theory of web service choreographies. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 1–16. Springer, Heidelberg (2008)

13. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and orchestration: A synergic approach for system design. In: Benatallah, B., Casati, F., Traverso, P. (eds.) ICSOC 2005. LNCS, vol. 3826, pp. 228–240. Springer, Heidelberg (2005)
14. Martens, A.: Analyzing web service based business processes. In: Cerioli, M. (ed.) FASE 2005. LNCS, vol. 3442, pp. 19–33. Springer, Heidelberg (2005)
15. Fu, X., Bultan, T., Su, J.: Analysis of interacting BPEL web services. In: Proc. WWW 2004, pp. 621–630. ACM, New York (2004)
16. Canal, C., Pimentel, E., Troya, J.M.: Compatibility and inheritance in software architectures. Sci. Comput. Program. 41(2), 105–138 (2001)
17. Puhlmann, F., Weske, M.: Interaction soundness for service orchestrations. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 302–313. Springer, Heidelberg (2006)
18. Lohmann, N.: Correcting deadlocking service choreographies using a simulation-based graph edit distance. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 132–247. Springer, Heidelberg (2008)
19. Fu, X., Bultan, T., Su, J.: Synchronizability of conversations among web services. IEEE Trans. Softw. Eng. 31(12), 1042–1055 (2005)
20. Reisig, W.: Elements of Distributed Algorithms: Modeling and Analysis with Petri Nets. Springer, Heidelberg (1998)
21. OMG: Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification. Technical report, Object Management Group, OMG (2006)