# Correcting Deadlocking Service Choreographies Using a Simulation-Based Graph Edit Distance

Niels Lohmann

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

**Abstract.** Many work has been conducted to analyze service choreographies to assert manyfold correctness criteria. While errors can be *detected* automatically, the *correction* of defective services is usually done manually. This paper introduces a graph-based approach to calculate the minimal edit distance between a given defective service and synthesized correct services. This edit distance helps to automatically fix found errors while keeping the rest of the service untouched. A prototypic implementation shows that the approach is applicable to real-life services.

**Keywords:** Choreographies, graph correction, correction of services, verification of services, service automata, operating guidelines, BPEL.

## 1 Introduction

In service-oriented computing [1], the correct interplay of distributed services is crucial to achieve a common goal. Choreographies [2] are a means to document and model the complex global interactions between services of different partners. BPEL4Chor [3] has been introduced to use BPEL [4] to describe and execute choreographies. Recently, a formal semantics for BPEL4Chor was introduced [5], offering tools and techniques to verify BPEL-based choreographies.

Whereas it is already possible to automatically *check* choreographies for deadlocks or to synthesize participant services [6], no work was conducted in supporting the *fixing* of existing choreographies. This is especially crucial, because fixing incorrect services is usually cheaper and takes less time than re-designing and implemeting a correct service from scratch. In addition, information on how to adjust an existing service can help the designers understand the error more easily compared to confronting them with a whole new synthesized service.

As the running example for this paper, consider the example choreography visualized in BPMN [7] in Fig. 1. It describes the interplay of a travel agency, a customer service, and an airline reservation system. The travel agency sends an offer to the client which either rejects it or books a trip. In the latter case, the travel agency orders a ticket at the airline service which either sends a confirmation or a refusal message to the customer. The choreography contains a design flaw as the customer service does not receive the refusal message. This leads to a deadlock in case the airline refuses the ticket order.

M. Dumas, M. Reichert, and M.-C. Shan (Eds.): BPM 2008, LNCS 5240, pp. 132–147, 2008.

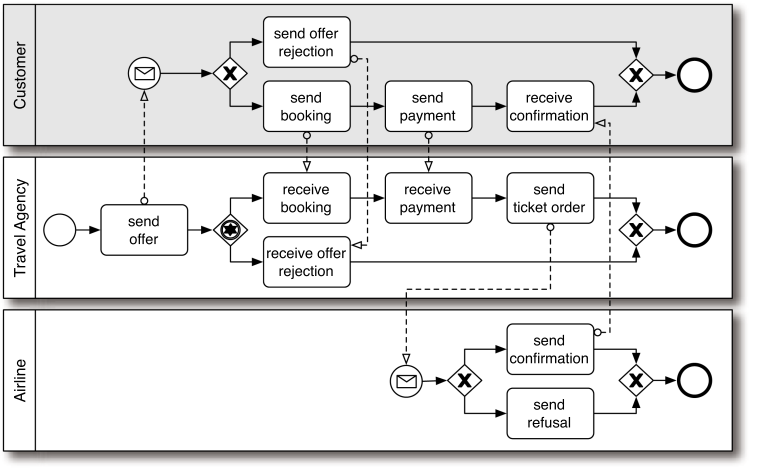


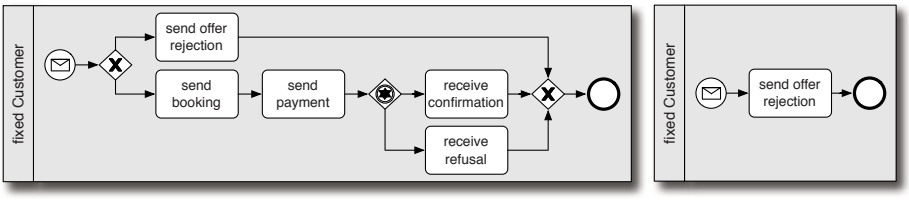

**Fig. 1.** Choreography between travel agency, airline, and customer. The choreography can deadlock, because the customer does not receive a refusal message from the airline.

This deadlock can be detected using state-of-the-art model checking tools which provide a trace to the deadlocking state. In the concrete example, a trace would be (send offer, receive offer, send booking, send payment[1], receive booking, receive payment, send ticket order, receive ticket order, send refusal). This trace, however, gives no information which service has to be changed in which manner to avoid the deadlock. Thus, an iteration of manual corrections followed by further deadlock checks is necessary to finally remove the deadlock. Though it is obvious how to correct the flawed example, the manual correction of choreographies of a larger number of more complex services is tedious, if not impossible.

Moreover, even for this simple choreography exists a variety of possibilities to fix the customer's service. Figure 2 depicts two possible corrections to avoid the deadlock. Though both services would avoid the choreography to deadlock, the service in Fig. 2(a) is to be preferred over that in Fig. 2(b) as it is "more similar" to the original service. Though this preference is psychological and is unlikely to be proven formally, the usage of similarities is widley accepted (cf. [8]). The tool chain presented in [6,5] synthesizes a participant service independently of an existing incorrect service and might produce correct, yet unintuitive results such as the service in Fig. 2(b).

The goal of this paper is to formalize, systematize, and to some extend automatize the fixing of choreographies as it has been illustrated above. We thereby combine existing work on characterizing all correctly interacting partners of a service with similarity measures and edit distances known in the field of graph correction. These approaches are recalled in Sect. 2 and 3. In Sect. 4, we define an edit distance that aims at finding the *most similar* service from the set of all fitting services. To support the modeler, we further derive the required edit

[1] We assume asynchronous (i. e., non-blocking) communication.




(a) add receipt of refusal message (b) delete booking branch

**Fig. 2.** Two possible corrections of the customer service to achieve deadlock freedom

actions needed to correct the originally incorrect service. In Sect. 5, we present experimental results conducted with a proof of concept implementation. Section 6 discusses related work. Finally, Sect. 7 is dedicated to a conclusion and gives directions for future research.

## 2 Service Models

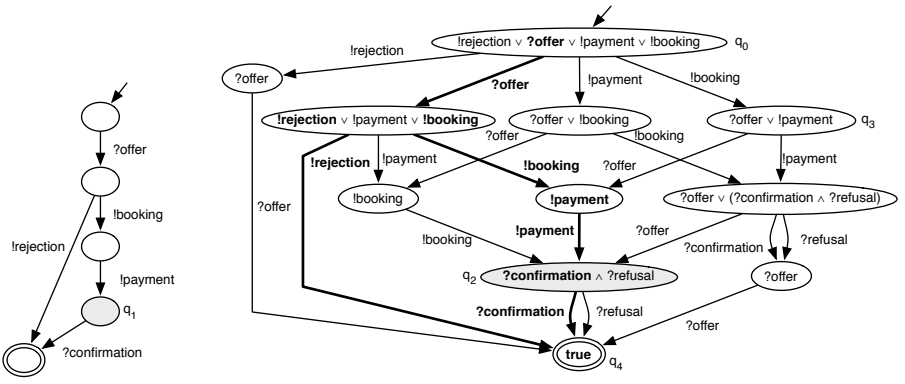
### 2.1 Service Automata and Operating Guidelines

To formally analyze services, a sound mathematical model is needed. In the area of workflows and services, Petri nets are a widely accepted formalism [9]. They combine a graphical notation with a variety of analysis methods and tools. For real-life service description languages such as BPEL or BPEL4Chor exists a feature-complete Petri net semantics [10,5]. To simplify the presentation, we abstract from the structure of a service and complex aspects such as data or fault handling, and focus on the external behavior (also known as the *business protocol*) of services in this paper. To this end, we use service automata [11] to model the external behavior services.[2]

A *service automaton* is a finite automaton with a set $Q$ of states, a set $F \subseteq Q$ of final states, an initial state $q_0 \in Q$, an interface $I$ for asynchronous message passing, and a partial transition function $\delta : Q \times I \to Q$. In this paper, we only consider deterministic service automata and require that final states are sink states; that is, have no outgoing transitions. For $\delta(q, a) = q'$ we also write $q \xrightarrow{a} q'$. Throughout this paper, we use $\mathcal{S}$ to denote service automata. We further assume that all services in this paper share a common interface $I$. This common interface can be achieved by joining all participants' interfaces.

Figure 3(a) depicts a service automaton modeling the external behavior of the customer service of Fig. 1. The edges are labeled with messages sent to (preceded with "!") or received from (preceded with "?") the environment: The interface of the service automaton is {!booking, ?confirmation, ?offer, !payment, !rejection}.

[2] Due to the close relationship (cf. [12]) between Petri nets and automata, there exist techniques to transform back and forth between the two formalisms.




(a) $\mathcal{S}_{\mathsf{customer}}$ (b) $\mathcal{O}_{\mathsf{agency} \oplus \mathsf{airline}}$

**Fig. 3.** A service automaton $\mathcal{S}_{\mathsf{customer}}$ modeling the customer service of Fig. 1 (a) and the operating guideline $\mathcal{O}_{\mathsf{agency} \oplus \mathsf{airline}}$ of the composition of travel agency and airline service (b). The service automaton does not match the OG, because $\mathsf{q}_2$'s formula is not satisfied.

As services are usually not considered in isolation, their interplay has to be taken into account in verification. A necessary correctness criterion is *controllability* [13]. A service $\mathcal{S}$ is controllable if there exists a partner service $\mathcal{S}'$ such that their composition $\mathcal{S} \oplus \mathcal{S}'$ (i. e., the choreography of the service and the partner) is free of deadlocks.

Controllability can be decided constructively: If a correctly interacting partner service for $\mathcal{S}$ exists, it can be automatically synthesized [13,6]. Furthermore, it has been proven that there exists one distinguished partner service $\mathcal{S}^*$ that is *most permissive*; that is, it simulates any other correctly interacting partner service. The converse does, however, not hold; not every simulated service is a correct partner service itself. To this end, the most permissive partner service can be annotated with Boolean formulae expressing which states and transitions can be omitted and which parts are mandatory. This annotated most permissive partner service is called an *operating guideline* (OG) [11]. We denote OGs with $\mathcal{O}$ and use $\varphi(q)$ to denote the Boolean formula annotated to state $q$ of the OG.

Figure 3(b) depicts the OG of the composition of the travel agency and the airline. The disjunction of the OG's initial $\mathsf{q}_0$ state means that a partner service must send a rejection, receive an offer, send a payment or send a booking in its initial state. This is possible due to asynchronous communication. The service automaton of Fig. 3(a) is simulated by the OG and fulfills all but one formula (satisfied literals are depicted bold in Fig. 3(b)). It does not satisfy the formula $\varphi(\mathsf{q}_2) = (\mathsf{?confirmation} \wedge \mathsf{?refusal})$ of the OG's state $\mathsf{q}_2$, because the service automaton does not receive a refusal message in the simulated state $\mathsf{q}_1$.

### 2.2 Fixing Deadlocking Choreographies

Consider a deadlocking choreography of $n$ participants, $\mathcal{S}_1 \oplus \cdots \oplus \mathcal{S}_n$. As mentioned earlier, a deadlock trace usually does not give sufficient information *how* to fix *which* service to achieve deadlock freedom. To find a candidate service that can be changed such that whole choreography is deadlock-free, we can perform the following steps:

- Firstly, we check for each service the necessary correctness criterion: If a service taken for itself is not controllable, then there exists no environment in which that service runs correctly — especially not the choreography under consideration. In that case, that service has to be radically overworked towards controllability, which is not topic of this paper.
- Secondly, we remove one participant, say $\mathcal{S}_i$. The resulting choreography $Chor_i = \mathcal{S}_1 \oplus \cdots \oplus \mathcal{S}_{i-1} \oplus \mathcal{S}_{i+1} \oplus \cdots \oplus \mathcal{S}_n$ can be considered as one large service with an interface. If it is controllable, then there exists a service $\mathcal{S}'_i$ which interacts deadlock-freely with the other participants of the choreography; that is, $Chor_i \oplus \mathcal{S}'_i$ is deadlock-free. In [5], a complete tool chain for this participant synthesis was presented for BPEL-based choreographies.

As motivated in the introduction, the mere replacement of $\mathcal{S}_i$ by $\mathcal{S}'_i$ is not desirable, because $\mathcal{S}'_i$ totally ignores the structure of $\mathcal{S}_i$ and might be very different to the original, yet incorrect service $\mathcal{S}_i$. Instead of synthesizing *any* fitting service (such as the service in Fig. 2(b)), we are interested in a corrected service that is most similar to $\mathcal{S}_i$. To this end, we can use the OG of $Chor_i$, because it characterizes the set of *all* fitting partners. Figure 4 illustrates this.

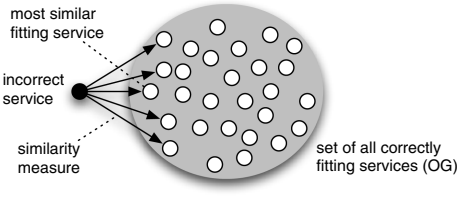



**Fig. 4.** The OG as characterization of all correct services can be used to find the most similar service

Beside the corrected services of Fig. 2, the OG characterizes 2002 additional (acyclic and deterministic) partner services.[3] Though all are correct, we are interested in the service that most similar to the incorrect customer service. Instead of iteratively check all candidates, we will define a similarity measure that exploits the OG's compact representation to efficiently find the desired service of Fig. 2(a).

[3] The set of cyclic or nondeterministic partner services might be infinite.

## 3 Graph Similarities

Graph similarities are widely used in many fields of computer science, for example for pattern recognition [14] or in bio informatics. Cost-based distance measures adapt the *edit distance* known from string comparison [15,16] to compare labeled graphs (e. g., [17]). They aim at finding the minimal number of modifications (i. e., adding, deleting, and modifying nodes and edges) needed to achieve a graph isomorphism.

Distance measures aiming at graph isomorphism have the drawback that they are solely based on the *structure* of the graphs. They focus on the syntax of the graphs rather than their semantics. When a graph (e. g., a service automaton) models the *behavior* of a system, similarity of graphs should focus on simulation of behavior rather than on a high structural similarity. Figure 5 illustrates that structural and behavioral similarity is not necessarily related.

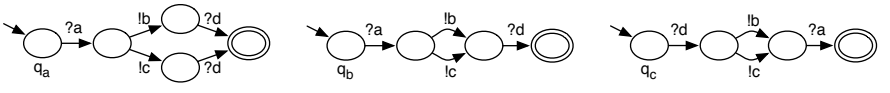



**Fig. 5.** Service automata (a) and (b) simulate each other, but have an unsimilar structure. Service automata (b) and (c) have a very similar structure, but rather unsimilar behavior.

Sokolsky et el. [18] address this problem (a similar approach is presented in [19]), motivated by finding computer viruses by comparing a program with a library of control flow graphs. In that setting, classical simulation is too strict, because two systems that are equal in all but one edge label *behave* very similar, but there exists no simulation relation between them. To this end, Sokolsky et al. introduce a *weighted quantitative simulation* function to compare states of two graphs. Whenever the two graphs cannot perform a transition with same labels, one graph performs a special stuttering step $\varepsilon$, which is similar to $\tau$-steps in stuttering bisimulation [20]. To "penalize" stuttering, a label similarity function assings low similarity between $\varepsilon$ and any other label. This label similarity function $L : (I \cup \{\varepsilon\}) \times (I \cup \{\varepsilon\}) \to [0, 1]$ assigns a value that expresses the similarity between the labels of the service automata[4] under consideration. For example, $L(\mathsf{?a}, \mathsf{?b})$ describes the similarity of a ?a-labeled transition of service automaton $\mathcal{S}_1$ and a ?b-labeled transition of service automaton $\mathcal{S}_2$. Furthermore, a discount factor $p \in [0, 1]$ describes the local importance of similarity compared to the similarity of successor states.

**Definition 1 (Weighted quantitative simulation, [18]).** *Let* $\mathcal{S}_1 = [Q_1, \delta_1, F_1, q_{0_1}, I]$, $\mathcal{S}_2 = [Q_2, \delta_2, F_2, q_{0_2}, I]$ *be service automata. A* weighted quantitative simulation *is a function* $S : Q_1 \times Q_2 \to [0, 1]$*, such that:*

[4] We adjusted the definitions of to service automata. The original definition in [18] bases on labeled directed graphs. We do not consider node labels in this paper.

$$S(q_1, q_2) = \begin{cases} 1, & \text{if } q_1 \in F_1, \\ (1-p) + \max(W_1(q_1, q_2), W_2(q_1, q_2)), & \text{otherwise,} \end{cases}$$

$$W_1(q_1, q_2) = \max_{q_2 \xrightarrow{b} q_2'} \Big( L(\varepsilon, b) \cdot S(q_1, q_2') \Big),$$

$$W_2(q_1, q_2) = \frac{p}{n} \cdot \sum_{q_1 \xrightarrow{a} q_1'} \max \left( L(a, \varepsilon) \cdot S(q_1', q_2), \max_{q_2 \xrightarrow{b} q_2'} (L(a, b) \cdot S(q_1', q_2')) \right),$$

*and $n$ is the number of edges leaving $q_1$.*

The weighted quantitative simulation function $S$ recursively compares the states from the two service automata and finds the maximal similar edges. Thereby, $W_1$ describes the similarity gain by stuttering of graph $\mathcal{S}_1$ and $W_2$ the tradeoff between simultaneous transitions of $\mathcal{S}_1$ and $\mathcal{S}_2$ and stuttering of graph $\mathcal{S}_2$. Both, the discount factor $p$ and the label similarity function $L$, can be chosen freely to adjust the result of the similarity algorithm. The choice of the parameters is, however, out of scope of this paper.

As an example, consider the service automata in Fig. 5 and assume a discount factor $p = 0.5$ and a label similarity function $L$ that assigns 1.0 to equal labels and 0.5 to any other label pair. Then $S(\mathsf{q_a}, \mathsf{q_b}) = 1.0$ (the weighted quantitative simulation is a generalization of the classical simulation) and $S(\mathsf{q_b}, \mathsf{q_c}) = 0.75$ which indicates the differences in the behaviors.

## 4 A Matching-Based Edit Distance

The algorithm to calculate weighted quantitative simulation can be used as a similarity measure for service automata or OGs, but has two drawbacks: Firstly, it is not an edit distance. It calculates a value that expresses the similarity between the service automata, but gives no information about the modification actions needed to *achieve* simulation. Secondly, it does not take formulae of the OG into account. Therefore, a high similarity between a service automaton and an OG would not guarantee deadlock freedom as the example of Fig. 3 demonstrates: The service automaton of the customer is perfectly simulated by the OG but the overall choreography deadlocks.

### 4.1 Simulation-Based Edit Distance

Before we consider the OG's formulae, we show how the similarity result of the algorithm of [18] can transformed into an edit distance. Given two states $q_1$ and $q_2$, Def. 1 determines the best simulation between the transitions of $q_1$ and $q_2$. In addition, one service automaton can stutter (i. e., remain in the same state). The weighted quantitative simulation function calculates the best label matching to maximize the similarity between the root nodes of the service automata. From the transition pairs belonging to the maximum, we can derive according edit actions (cf. Table 1).

**Table 1.** Deriving edit actions from transition pairs of Def. 1

| transition of $\mathcal{S}_1$ | transition of $\mathcal{S}_2$ | resulting edit action | similarity |
|---|---|---|---|
| $\mathsf{a}$ | $\mathsf{a}$ | keep transition $\mathsf{a}$ | $L(\mathsf{a},\mathsf{a})$ |
| $\mathsf{a}$ | $\mathsf{b}$ | modify transition $\mathsf{a}$ to $\mathsf{b}$ | $L(\mathsf{a},\mathsf{b})$ |
| $\mathsf{a}$ | $\varepsilon$ (stutter) | delete transition $\mathsf{a}$ | $L(\mathsf{a},\varepsilon)$ |
| $\varepsilon$ (stutter) | $\mathsf{a}$ | insert transition $\mathsf{a}$ | $L(\varepsilon,\mathsf{a})$ |

These edit actions define basic edit actions whose similarity is determined by the edge similarity function $L$. To simplify the representation of a large number of edit actions, the basic edit actions may be grouped to macros to express more complex operations such as swapping or moving of edges and nodes, duplicating of subgraphs, or partial unfolding of loops.

The simulation-based edit distance does not respect the OG's formulae. One possibility to achieve a matching would be to first calculate the most similar simulating service using the edit distance for Def. 1 and then to simply add and remove all nodes and edges necessary in a second step. Using the weighted quantitative simulation function of Def. 1, the resulting edit actions (cf. Table 1) simply inserts or removes edges to present nodes rather than to new nodes. This approach does in general not work to achieve matching with an OG. See Fig. 6 for a counterexample. However, also the insertion of nodes would not determine the most similar partner service, because this may result in sub-optimal solutions as Fig. 7 illustrates.

### 4.2 Combining Formula-Checking and Graph Similarity

Due to the suboptimal results achieved by a-posteriori formula satisfaction by node insertion, we need to modify the algorithm of [18] not to statically take the outgoing transitions of an OG's state into account, but also check any formula-fulfilling subset of outgoing transitions. Therefore, we need some additional definitions to base formula satisfaction and to cover the dynamic presence of OG transitions.

**Definition 2 (Satisfying label set, label permutation).** *Let* $\mathcal{S} = [Q_\mathcal{S}, \delta_\mathcal{S}, F_\mathcal{S}, q_{0_\mathcal{S}}, I]$ *be a service automaton and* $\mathcal{O} = [Q_\mathcal{O}, \delta_\mathcal{O}, F_\mathcal{O}, q_{0_\mathcal{O}}, I]$ *an OG, and let* $q_1 \in Q_\mathcal{S}$ *and* $q_2 \in Q_\mathcal{O}$.

- *Define* $Sat(\varphi(q_2)) \subseteq \mathcal{P}(I \cap \{b \mid \exists q_2' \in Q_\mathcal{O} : q_2 \xrightarrow{b} q_2'\})$ *to be the set of all sets of labels of transitions leaving* $q_2$ *that satisfy formula* $\varphi$ *of state* $q_2$.
- *For* $\beta \in Sat(\varphi(q_2))$, *define* $perm(q_1, q_2, \beta) \subsetneq \big((I \cup \{\varepsilon\}) \times (I \cup \{\varepsilon\})\big)$ *to be a label permutation of* $q_1$, $q_2$ *and* $\beta$ *such that:*
  - *(a) if* $q_1 \xrightarrow{a} q_1'$, *then* $(a, c) \in perm(q_1, q_2, \beta)$ *for a label* $c \in \beta \cup \{\varepsilon\}$,
  - *(b) if* $q_2 \xrightarrow{b} q_2'$ *and* $b \in \beta$, *then* $(d, b) \in perm(q_1, q_2, \beta)$ *for a label* $d \in I \cup \{\varepsilon\}$,
  - *(c)* $(\varepsilon, \varepsilon) \notin perm(q_1, q_2, \beta)$, *and*
  - *(d) if* $(a, b) \in perm(q_1, q_2, \beta)$, *then* $(a, c), (d, b) \notin perm(q_1, q_2, \beta)$ *for all labels* $c \in \beta \cup \{\varepsilon\}$ *and all labels* $d \in I \cup \{\varepsilon\}$.
- *Define* $Perms(q_1, q_2, \beta)$ *to be the set of all label permutations of* $q_1$, $q_2$ *and* $\beta$.

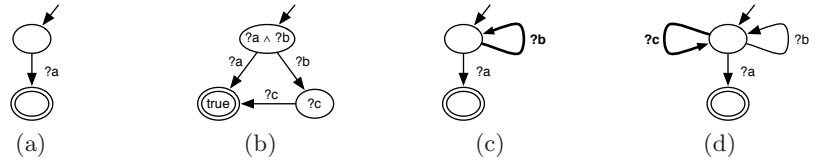



**Fig. 6.** Matching cannot be achieved solely by transition insertion. The service automaton (a) does not match with the OG (b) because of a missing ?b-branch. In service automaton (c), a loop edge was inserted. However, the state reached by ?b in the OG requires a ?c-branch to be present. After inserting this edge (d), the resulting service automaton is not simulated by the OG (b).

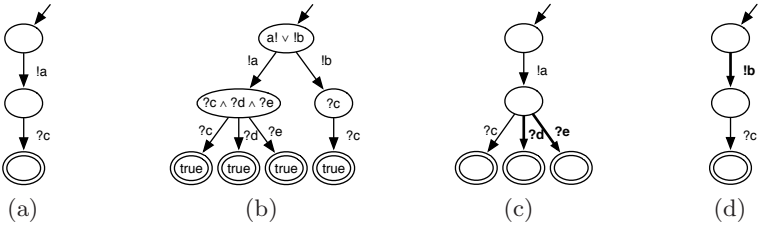



**Fig. 7.** Adding states to a simulating service automaton may yield sub-optimal results. The service automaton (a) does not match with the OG (b), because the formula (?c∧?d∧?e) is not satisfied. The OG, however, perfectly simulates the service automaton (a), and adding two edges achieves matching (c). However, changing the edge label of (a) from !a to !b also achieves matching, but only requires a single edit action (d).

The set $Sat$ consists of all sets of labels that fulfill a state's formula. For example, consider the OG in Fig. 3(b): For state $\mathsf{q}_2$ of the OG $\mathcal{O}_{\mathsf{agency} \oplus \mathsf{airline}}$, we have $Sat(\varphi(\mathsf{q}_2)) = \{\{?\mathsf{confirmation}, ?\mathsf{refusal}\}\}$. Likewise, $Sat(\varphi(\mathsf{q}_3)) = \{\{?\mathsf{offer}\}, \{!\mathsf{payment}\}, \{?\mathsf{offer}, !\mathsf{payment}\}\}$.

The set $Perms$ consists of all permutations of outgoing edges of two states. In a permutation, each outgoing edge of a state of the service automaton has to be present as first element of a pair (a), each outgoing edge of a state of the OG that is part of the label set $\beta$ has to be present as second element of a pair (b). As the number of outgoing edges of both states may be different, $\varepsilon$-labels can occur in the pairs, but no pair $(\varepsilon, \varepsilon)$ is allowed (c). Finally, each edge is only allowed to occur once in a pair (d).

For $\beta = \{?\mathsf{confirmation}, ?\mathsf{refusal}\}$ and state $\mathsf{q}_1$ of the service automaton $\mathcal{S}_1$ in Fig. 3(a), $\{(?\mathsf{confirmation}, ?\mathsf{confirmation}), (\varepsilon, ?\mathsf{refusal})\}$ is one of the permutations in $Perms(\mathsf{q}_1, \mathsf{q}_2, \beta)$. Another permutation is $\{(?\mathsf{confirmation}, ?\mathsf{refusal}), (\varepsilon, ?\mathsf{confirmation})\}$. The permutations can be interpreted like the label pairs of the simulation edit distance: $(?\mathsf{confirmation}, ?\mathsf{confirmation})$ describes a keeping of ?confirmation, $(?\mathsf{confirmation}, ?\mathsf{refusal})$ describes changing ?confirmation to ?refusal, and $(\varepsilon, ?\mathsf{refusal})$ the insertion of a ?refusal transition. The insertion and deletion has to be adapted to avoid incorrect or sub-optimal results (see Fig. 6–7).

**Definition 3 (Subgraph insertion, subgraph deletion).** *Let* $\mathcal{S} = [Q_\mathcal{S}, \delta_\mathcal{S}, F_\mathcal{S}, q_{0_\mathcal{S}}, I]$ *be a service automaton and* $\mathcal{O} = [Q_\mathcal{O}, \delta_\mathcal{O}, F_\mathcal{O}, q_{0_\mathcal{O}}, I]$ *an OG. Define*

$$ins(q_2) = \begin{cases} 1, & \text{if } q_2 \in F_\mathcal{O}, \\ (1-p) + \max\limits_{\beta \in Sat(\varphi(q_2))} \dfrac{p}{|\beta|} \cdot \sum\limits_{b \in \beta} L(\varepsilon, b) \cdot ins(\delta_\mathcal{O}(q_2, b)), & \text{otherwise,} \end{cases}$$

$$del(q_1) = \begin{cases} 1, & \text{if } q_1 \in F_\mathcal{S}, \\ (1-p) + \dfrac{p}{n} \cdot \sum\limits_{q_1 \xrightarrow{a} q_1'} L(a, \varepsilon) \cdot del(q_1'), & \text{otherwise,} \end{cases}$$

*where* $n$ *is the number of outgoing edges of* $q_1$*.*

Function $ins(q_2)$ calculates the insertion cost of the optimal subgraph of the OG $\mathcal{O}$ beginning at $q_2$ which fulfills the formulae. Likewise, $del(q_1)$ calculates the cost of deletion of the whole subgraph of the service automaton $\mathcal{S}$ from state $q_1$. Both functions only depend on one of the graphs; that is, *ins* and *del* can be calculated independently from the service automaton and the OG, respectively. Definition 3 actually does not insert or delete nodes, but only calculates the similarity value of the resulting subgraphs. Only this similarity is needed to find the most similar partner service and the actual edit actions can be easily derived from the state from which nodes are inserted or deleted (cf. Table 1).

With Def. 2 describing means to respect the OG's formulae and Def. 3 coping with insertion and deletion, we can finally define the weighted quantitative matching function:

**Definition 4 (Weighted quantitative matching).** *Let* $\mathcal{S} = [Q_\mathcal{S}, \delta_\mathcal{S}, F_\mathcal{S}, q_{0_\mathcal{S}}, I]$ *be a service automaton and* $\mathcal{O} = [Q_\mathcal{O}, \delta_\mathcal{O}, F_\mathcal{O}, q_{0_\mathcal{O}}, I]$ *an OG. A* weighted quantitative matching *is a function* $M : Q_\mathcal{S} \times Q_\mathcal{O} \to [0, 1]$*, such that:*

$$M(q_1, q_2) = \begin{cases} 1, & \text{if } (q_1 \in F_\mathcal{S} \wedge q_2 \in F_\mathcal{O}), \\ (1-p) + W_1(q_1, q_2), & \text{otherwise,} \end{cases}$$

$$W_1(q_1, q_2) = \max_{\beta \in Sat(\varphi(q_2))} \max_{P \in Perms(q_1, q_2, \beta)} \frac{p}{|P|} \cdot \sum_{(a,b) \in P} W_2(q_1, q_2, a, b),$$

$$W_2(q_1, q_2, a, b) = \begin{cases} L(a, b) \cdot M(\delta_\mathcal{S}(q_1, a), \delta_\mathcal{O}(q_2, b)), & \text{if } (a \neq \varepsilon \wedge b \neq \varepsilon), \\ L(\varepsilon, b) \cdot ins(\delta_\mathcal{O}(q_2, b)), & \text{if } a = \varepsilon, \\ L(a, \varepsilon) \cdot del(\delta_\mathcal{S}(q_1, a)), & \text{otherwise.} \end{cases}$$

The weighted quantitative matching function is similar to the weighted quantitative simulation function (Def. 1). It recursively compares the states of the service automaton and the OG, but instead of statically taking the OG's edges into consideration, it uses the formulae and checks all satisfying subsets ($W_1$). Additionally, $W_2$ organizes the successor states determined by the labels $a$ and $b$, or the insertion or deletion.

### 4.3 Matching-Based Edit Distance

Again, we can straight-forwardly extend the weighted quantitative matching function towards an edit distance, because the permutations give information how to modify the graph. Keeping and modification of transitions is handled as in Table 1, whereas adding and deletion of nodes can be derived from Def. 3. In fact, the weighted quantitative matching function is not a classical distance. It expresses the similarity between a service automaton and an OG (i. e., a characterization of many service automata) and is hence not symmetric. We still use the term "edit distance" to express the concept of a similarity measure from which edit actions can be derived.

Consider the example from Fig. 3. During the calculation of $M(\mathsf{q}_1, \mathsf{q}_2)$, the permutation $\{(\mathsf{?confirmation}, \mathsf{?confirmation}), (\varepsilon, \mathsf{?refusal})\}$ is considered. The first label pair denotes that the ?confirmation transition is kept unmodified. The second label pair denotes an insertion of a ?refusal transition. The value of this insertion is defined by

$$\begin{aligned} L(\varepsilon, \mathsf{?refusal}) \cdot ins(\delta_{\mathcal{O}_{\mathsf{agency} \oplus \mathsf{airline}}}(\mathsf{q}_2, \mathsf{?refusal})) &= L(\varepsilon, \mathsf{?refusal}) \cdot ins(\mathsf{q}_4) \\ &= L(\varepsilon, \mathsf{?refusal}) \end{aligned}$$

and only depends on the similarity function $L$.

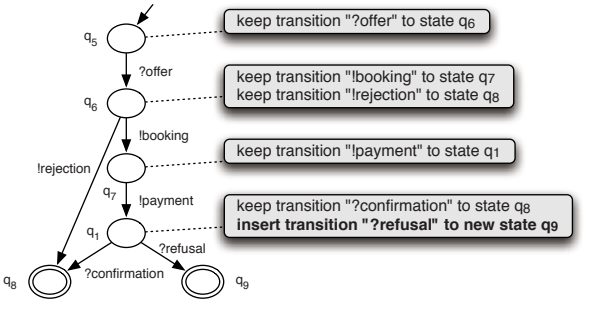



**Fig. 8.** Matching-based edit distance applied to the customer's service

Figure 8 shows the result of the application of the matching-based edit distance to the service automaton of Fig 3(a). The states are annotated with edit actions. The service automaton was automatically generated from a BPEL process and the state in which a modification has to be made can be mapped back to the original BPEL activity. In the example, a `receive` activity has to be replaced by a `pick` activity with an additional `onMessage` branch to receive the refusal message.

## 5 Complexity Considerations and Experimental Results

The original simulation algorithm of [18] to calculate a weighted quantitative simulation between two service automata $\mathcal{S}_1$ and $\mathcal{S}_2$ (cf. Def. 1) needs to check

$O(|Q_{\mathcal{S}_1}| \cdot |Q_{\mathcal{S}_2}|)$ state pairs. The extension to calculate the matching between a service automaton $\mathcal{S}$ and an OG $\mathcal{O}$ (cf. Def. 4) takes the OG's formulae and the resulting label permutations into consideration. The length of the OG's formulae is limited by the maximal degree of the nodes which again is limited by the interface $I$. Thus, for each state pair, at most $2^{|I|}$ satisfying assignments have to be considered. The number of permutations is again limited by the maximal node degree such that at most $|I|!$ permutations have to be considered for each state pair and assignment. This results in $O(|Q_{\mathcal{S}}| \cdot |Q_{\mathcal{O}}| \cdot 2^{|I|} \cdot |I|!)$ comparisons.

Though the extension towards a formula-checking edit distance yields a high worst-case complexity, OGs of real-life services tend to have quite simple formulae, a rather small interface (compared to the number of states), and a low node degree. As a proof of concept, we implemented the edit distance in a prototype.[5] It takes an acyclic deterministic service automaton and an acyclic OG[6] as input and calculates the edit actions necessary to achieve a matching with the OG. The prototype exploits the fact that a lot of subproblems overlap, and uses dynamic programming techniques [21] to cache and reuse intermediate results which significantly accelerates the runtime. We evaluated the prototype with models of some real-life services. In most cases, the edit distance could be calculated within few seconds. The experiments were conducted on a 2.16 GHz notebook. Memory consumption is not listed as it never exceeded 10 MB. Table 2 summarizes the results.

**Table 2.** Experimental results

| service | interface | states SA | states OG | search space | time (s) |
|---|---|---|---|---|---|
| Online Shop | 16 | 222 | 153 | $10^{2033}$ | 4 |
| Supply Order | 7 | 7 | 96 | $10^{733}$ | 1 |
| Customer Service | 9 | 104 | 59 | $10^{108}$ | 3 |
| Internal Order | 9 | 14 | 512 | $> 10^{4932}$ | 195 |
| Credit Preparation | 5 | 63 | 32 | $10^{36}$ | 2 |
| Register Request | 6 | 19 | 24 | $10^{25}$ | 0 |
| Car Rental | 7 | 49 | 50 | $10^{144}$ | 6 |
| Order Process | 8 | 27 | 44 | $10^{222}$ | 0 |
| Auction Service | 6 | 13 | 395 | $10^{12}$ | 0 |
| Loan Approval | 6 | 15 | 20 | $10^{17}$ | 0 |
| Purchase Order | 10 | 137 | 168 | $> 10^{4932}$ | 391 |

The first seven services are derived from BPEL processes of a German consulting company; the last four services are taken from the current BPEL specification [4]. The services were translated into service automata using the compiler BPEL2oWFN.[7] For these service automata, the OGs were calculated using the tool Fiona.[8] For some services, a partner service was already available; for the other services, we synthesized a partner service with Fiona. As we can

[5] Available at `http://service-technology.org/rachel`.
[6] Operating guidelines are deterministic by construction.
[7] Available at `http://service-technology.org/bpel2owfn`.
[8] Available at `http://service-technology.org/fiona`.

see, the services' interfaces are rather small compared to their number of states. It is worth mentioning that the complexity of the matching is independent of the fact whether the service automaton matches the OG or not. We used existing partner services in the case study to process services of realistic size.

Column "search space" of Table 2 lists the number of acyclic deterministic services characterized by the OG. All these services are correct partner services and have to be considered when finding the most similar service. The presented algorithm exploits the compact representation of the OG and allows to efficiently find the most similar service from more than $10^{2000}$ candidates.

For most services, the calculation only takes a few seconds. The "Internal Order" and "Purchase Order" services are exceptions. The OGs of these services have long formulae with a large number of satisfying assignments (about ten times larger than those of the other services) yielding a significantly larger search space. Notwithstanding the larger calculation time, the service fixed by the calculated edit actions is correct by design, and the calculation time is surely an improvement compared to iterative manual correction.

## 6 Related Work

The presented matching edit distance is related to several aspects of current research in many areas of computer science:

*Automated debugging.* In the field of model checking, the explanation of errors by using distance metrics (cf. [8]) has received a lot of attention. Compared to the approach presented in this paper, these works focus on the explanation and location of single errors in classical C (i e., low-level) programs. The derived information is used to support the debugging of an erroneous program.

*Service matching.* Many works exists to discover a similar partner service. An approach to match BPEL processes using an algorithm based on subgraph isomorphism is presented in [22]. Other approaches such as [23,24] use ontologies and take the semantics of activities into account, but do not focus much on the behavior or message exchange. In [25], the behavior of a service is represented as a language of traces which allows for string edit distances to compare services. This approach, however, cannot be used in the setting of communicating services where the moment of branching is crucial to avoid deadlocks.

*Service similarity and versioning.* The change management of business processes and services is subject of many recent works. An overview of what can differ between otherwise similar services is given in [26,27]. The reported differences go beyond the behavioral level and also take authorization aspects under consideration. [28] gives an overview of frequent change patterns occurring in the evolution of a business process model. Beside the already mentioned basic operations (adding, changing and removing of edges or nodes), complex operations such as extracting sub processes are presented. With a *version preserving graph*, a technique to represent different versions of a process model is introduced in [29]. This technique was made independent of a change log in [30]. Again, versioning relies on the structure of the model rather than on its behavior.

*Service mediation.* Instead of changing a service to achieve deadlock freedom in a choreography, it would also be possible to use a service mediator (sometimes called adapter) to fix a choreography (e. g., [31,32]). Service mediation is rather suited to fit existing services, whereas our approach aims at supporting the design and modelling phase of a service choreography. Still, a mediator between the customer service on the one hand and the travel agency and the airline service on the other hand (cf. Fig. 1) would have to receive the airline's refusal message and create a confirmation message for the customer which is surely unintended. Furthermore, several service mediation approaches such as [33] assume total perception of the participants' internal states during runtime.

The difference between all mentioned related approaches and the setting of this paper is that these approaches either focus on low-level programs or mainly aim at finding *structural* (certainly not simulation-based) differences between *two* given services and are therefore not applicable to find the most similar service from a large set (cf. Table 2) of candidates.

## 7 Conclusion and Future Work

We presented an edit distance to compute the edit actions necessary to correct a faulty service to interact deadlock-freely in a choreography. The edit distance (i. e., the actions needed to fix the service) can be automatically calculated using a prototypic implementation. Together with translations from [10] and to [34] BPEL processes and the calculation of the characterization of all correct partner services (the operating guideline) [6,11], a continuous tool chain to analyze and correct BPEL-based choreographies is available. As the edit distance itself bases on service automata, it can be easily adapted to other modeling languages such as UML activity diagrams [35] or BPMN [7] using Petri net or automaton-based formalisations.

However, a lot of questions still remain open. First of all, the choice *which* service causes the deadlock and hence needs to be fixed is not always obvious and needs further investigation. For instance, the choreography of Fig. 1 could also have been fixed by adjusting the airline service. Another aspect to be considered in future research is the choice of the *cost function* used in the algorithm, because it is possible to set different values for any transition pairs. Semantic information on message contents (e. g., derived from an ontology) and relationships between messages can be incorporated to refine the correction. For example, the insertion of the receipt of a confirmation message can be penalized less than the insertion of sending an additional payment message.

Another important field of research is to further increase the performance of the implementation by an early omission of suboptimal edit actions. For instance, heuristic guidance metrics such as used in the A* algorithm [36] may greatly improve runtime performance. Finally, a translation of the matching edit distance of Def. 4 into a linear optimization problem [37] may also help to cope with cyclic and nondeterministic services.

**Acknowledgements.** The author wishes to thank Oleg Sokolsky for the sources of his implementation, Christian Stahl for his feedback on an earlier version of this paper, and the anonymous referees for their valuable comments. Niels Lohmann is funded by the DFG project "Operating Guidelines for Services" (WO 1466/8-1).

## References

1. Papazoglou, M.P.: Agent-oriented technology in support of e-business. Commun. ACM 44(4), 71–77 (2001)
2. Dijkman, R., Dumas, M.: Service-oriented design: A multi-viewpoint approach. IJCIS 13(4), 337–368 (2004)
3. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for modeling choreographies. In: ICWS 2007, pp. 296–303. IEEE, Los Alamitos (2007)
4. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0, April 11, 2007. OASIS Standard, OASIS (2007)
5. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and participant synthesis. In: WS-FM 2007. LNCS, vol. 4937, pp. 46–60. Springer, Heidelberg (2008)
6. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting BPEL processes. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 17–32. Springer, Heidelberg (2006)
7. OMG: Business Process Modeling Notation (BPMN) Specification. Final Adopted Specification, Object Management Group (2006), `http://www.bpmn.org`
8. Groce, A., Chaki, S., Kroening, D., Strichman, O.: Error explanation with distance metrics. STTT 8(3), 229–247 (2006)
9. van der Aalst, W.M.P.: The application of Petri nets to workflow management. Journal of Circuits, Systems and Computers 8(1), 21–66 (1998)
10. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In: WS-FM 2007. LNCS, vol. 4937, pp. 77–91. Springer, Heidelberg (2008)
11. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 321–341. Springer, Heidelberg (2007)
12. Badouel, E., Darondeau, P.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) APN 1998. LNCS, vol. 1491, pp. 529–586. Springer, Heidelberg (1998)
13. Schmidt, K.: Controllability of open workflow nets. In: EMISA 2005. LNI, vol. P-75, pp. 236–249, GI (2005)
14. Sanfeliu, A., Fu, K.S.: A distance measure between attributed relational graphs for pattern recognition. IEEE Trans. on SMC 13(3), 353–362 (1983)
15. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Dokl. 10(8), 707–710 (1966)
16. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. J. ACM 21(1), 168–173 (1974)
17. Tsai, W., Fu, K.: Error-correcting isomorphisms of attributed relational graphs for pattern analysis. IEEE Trans. on SMC 9(12), 757–768 (1979)
18. Sokolsky, O., Kannan, S., Lee, I.: Simulation-based graph similarity. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS, vol. 3920, pp. 426–440. Springer, Heidelberg (2006)
19. Nejati, S., Sabetzadeh, M., Chechik, M., Easterbrook, S.M., Zave, P.: Matching and merging of statecharts specifications. In: ICSE, pp. 54–64. IEEE Computer Society, Los Alamitos (2007)

20. Namjoshi, K.S.: A simple characterization of stuttering bisimulation. In: Ramesh, S., Sivakumar, G. (eds.) FST TCS 1997. LNCS, vol. 1346, pp. 284–296. Springer, Heidelberg (1997)
21. Bellman, R.: Dynamic Programming. Princeton University Press, Princeton (1957)
22. Corrales, J.C., Grigori, D., Bouzeghoub, M.: BPEL processes matchmaking for service discovery. In: Meersman, R., Tari, Z. (eds.) OTM 2006. LNCS, vol. 4275, pp. 237–254. Springer, Heidelberg (2006)
23. Wu, J., Wu, Z.: Similarity-based Web service matchmaking. In: IEEE SCC, pp. 287–294. IEEE Computer Society, Los Alamitos (2005)
24. Bianchini, D., Antonellis, V.D., Melchiori, M.: Evaluating similarity and difference in service matchmaking. In: EMOI-INTEROP. CEUR Workshop Proceedings, CEUR-WS.org, vol. 200 (2006)
25. Günay, A., Yolum, P.: Structural and semantic similarity metrics for Web service matchmaking. In: Psaila, G., Wagner, R. (eds.) EC-Web 2007. LNCS, vol. 4655, pp. 129–138. Springer, Heidelberg (2007)
26. Dijkman, R.M.: A classification of differences between similar BusinessProcesses. In: EDOC, pp. 37–50. IEEE Computer Society, Los Alamitos (2007)
27. Dijkman, R.: Diagnosing differences between business process models. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, pp. 132–147. Springer, Heidelberg (2008)
28. Weber, B., Rinderle, S., Reichert, M.: Change patterns and change support features in process-aware information systems. In: Krogstie, J., Opdahl, A., Sindre, G. (eds.) CAiSE 2007 and WES 2007. LNCS, vol. 4495, pp. 574–588. Springer, Heidelberg (2007)
29. Zhao, X., Liu, C.: Version management in the business process change context. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) BPM 2007. LNCS, vol. 4714, pp. 198–213. Springer, Heidelberg (2007)
30. Küster, J.M., Gerth, C., Förster, A., Engels, G.: Detecting and resolving process model differences in the absence of a change log. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 244–260. Springer, Heidelberg (2008)
31. Brogi, A., Popescu, R.: Automated generation of BPEL adapters. In: Dan, A., Lamersdorf, W. (eds.) ICSOC 2006. LNCS, vol. 4294, pp. 27–39. Springer, Heidelberg (2006)
32. Dumas, M., Spork, M., Wang, K.: Adapt or perish: Algebra and visual notation for service interface adaptation. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 65–80. Springer, Heidelberg (2006)
33. Nezhad, H.R.M., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: WWW 2007, pp. 993–1002. ACM, New York (2007)
34. Lohmann, N., Kleine, J.: Fully-automatic translation of open workflow net models into simple abstract BPEL processes. In: Modellierung 2008. LNI, vol. P-127, pp. 57–72, GI (2008)
35. OMG: Unified Modeling Language (UML), Version 2.1.2. Technical report, Object Management Group (2007), `http://www.uml.org`
36. Hart, P.E., Nilsson, N.J., Raphael, B.: A formal basis for the heuristic determination of minimum cost paths in graphs. IEEE Trans.Syst. Sci. and Cybernetics SSC-4(2), 100–107 (1968)
37. Schrijver, A.: Theory of Linear and Integer Programming. John Wiley & sons, Chichester (1998)