

# Deciding Substitutability of Services with Operating Guidelines

Christian Stahl<sup>1,2,\*</sup>, Peter Massuthe<sup>1,2</sup>, and Jan Bretschneider<sup>1</sup>

<sup>1</sup> Humboldt-Universität zu Berlin, Institut für Informatik,  
Unter den Linden 6, 10099 Berlin, Germany

{stahl,massuthe,bretschn}@informatik.hu-berlin.de

<sup>2</sup> Department of Mathematics and Computer Science  
Technische Universiteit Eindhoven

P.O. Box 513, 5600 MB Eindhoven, The Netherlands

**Abstract.** Deciding whether a service  $S$  can be substituted by another service  $S'$  is an important problem in practice and one of the research challenges in service-oriented computing. In this paper, we define three substitutability notions for services. *Accordance* specifies that  $S'$  cooperates with at least the environments that  $S$  cooperates with.  $S$  and  $S'$  are *equivalent* if they cooperate with the same environments. To guarantee that  $S'$  cooperates with a fixed subset of environments that  $S$  cooperates with, the notion of *restriction* can be used. For each substitutability notion we present a decision algorithm. To this end we apply the concept of an *operating guideline* of a service as an abstract representation of all environments the service cooperates with.

**Key words:** Open nets, Operating guidelines, Service substitutability

## 1 Introduction

In the paradigm of service-oriented computing (SOC) [1], a service serves as a building block for designing flexible business processes by composing multiple services. Such a (composed) service is subject to changes. There may hardly ever be a total renewal or upgrade of the overall service. Instead, individual services will be replaced by better ones, because the service was too expensive or some new functionality has been added, for instance. *Service substitutability*, that is, deciding whether a service can be substituted by another service, is one of the most notable SOC research challenges [2].

Obviously, a service  $S$  can be substituted by another service  $S'$  if no environment can distinguish them, that is, they are equivalent. In practice, however, more flexible notions than equivalence are relevant as well. In general, substituting  $S$  by  $S'$  either should *gain* or *preserve* properties of the overall service.

In order to guarantee that substituting  $S$  by  $S'$  indeed gains and/or preserves specific properties, support of formal methods is needed. To this end we need

---

\* Funded by the DFG project “Substitutability of Services” (RE 834/16-1).

to characterize different properties of substitutability, resulting in different substitutability notions. In the next step, we have to develop algorithms to decide substitutability for each notion.

In this paper, we restrict ourselves to the service protocol, that is, to the *behavior* of a service, and abstract from other important aspects like quality of service and semantics. As our formal model we use *open nets*, a special class of Petri nets. An open net has an interface for communication with other open nets via asynchronous message passing. To meet different application scenarios that are relevant in practice we introduce three *substitutability notions*: accordance ( $S'$  cooperates with at least every environment  $S$  cooperates with), equivalence ( $S$  and  $S'$  cooperate with the same environments), and restriction ( $S'$  cooperates with at least a fixed subset of environments  $S$  cooperates with). Furthermore, a property-preserving substitutability notion is derived which is more fine-grained than restriction. For each such notion we present a decision algorithm based on the concept of an *operating guideline* as an abstract representation of all environments a given service can cooperate with. Operating guidelines have been suggested to support service discovery so far. In this paper, we show that operating guidelines are well-suited for deciding substitutability of services, too. To this end we use known results, extend some notions, and also provide new results on operating guidelines.

The remainder of this paper is structured as follows. Sections 2 and 3 present the preliminaries. There, we recall our service models, open nets and service automata, as well as operating guidelines. Then, in Sect. 4 we introduce the notion of accordance. Restriction is explained in Sect. 5. From accordance and restriction we derive in Sect. 6 two further substitutability notions. Related work is discussed in Sect. 7 and finally, conclusions are drawn in Sect. 8.

## 2 Service Models

In this section, we introduce *open nets*, a special class of Petri nets, as a formal model for services and *service automata* as a technique to analyze the interaction behavior of open nets. We will show that an open net can easily be translated into a service automaton and vice versa, so we can consider our analysis questions on both models alike.

### 2.1 Open Nets

We assume the usual definition of a (place/transition) *Petri net*  $N = [P, T, F]$  (see [3], for instance) and use the standard notation to denote the *preset* and *postset* of a place or a transition:  $\bullet x = \{y \mid (y, x) \in F\}$  and  $x^\bullet = \{y \mid (x, y) \in F\}$ .

A *marking* of a Petri net  $N$  is a mapping  $m : P \rightarrow \mathbb{N}$ . We use a *multiset* notation to denote markings and write  $m = [p_1, p_1, p_2]$  for a marking  $m$  with  $m(p_1) = 2$ ,  $m(p_2) = 1$ , and  $m(p) = 0$  for all  $p \in P \setminus \{p_1, p_2\}$ . If  $Q \supseteq P$ , a marking  $m : P \rightarrow \mathbb{N}$  extends canonically to  $m : Q \rightarrow \mathbb{N}$  by  $m(p) = 0$  for each  $p \in Q \setminus P$ .

*Open nets* were introduced in [4] using the term “open workflow nets”. Open nets are a special class of Petri nets and can be seen as a generalized version of van der Aalst’s workflow nets [5], which have been proven successful for the modeling of business processes and workflows. As a substantial difference, an open net has an *interface* that consists of a set of *input places* and a set of *output places* for asynchronous communication with an environment. This idea is based on the module concept for Petri nets which was proposed by Kindler [6]. Suitability of open nets for modeling services has been proven through an implemented translation (see [7], for instance) from the industrial service description language WS-BPEL [8] into open nets.

As a global name space, we assume a set  $\mathcal{MC}$  of *message channels* given. For technical reasons, we require that the special symbols  $\tau$  (representing a non-communicating step) and *final* (used to denote final states) are not in  $\mathcal{MC}$ .

**Definition 1 (Open net).**

An open net  $N = [P, P_{in}, P_{out}, T, F, m_0, \Omega]$  consists of a Petri net  $[P, T, F]$  together with

- two disjoint sets  $P_{in} \subseteq (P \cap \mathcal{MC})$  of input places such that  $\bullet p_{in} = \emptyset$  for all  $p_{in} \in P_{in}$  and  $P_{out} \subseteq (P \cap \mathcal{MC})$  of output places such that  $p_{out} \bullet = \emptyset$  for all  $p_{out} \in P_{out}$ ,
- a distinguished initial marking  $m_0$ , and
- a set  $\Omega$  of final markings such that no transition of  $N$  is enabled at any  $m \in \Omega$ .

Let  $P_{io} = P_{in} \cup P_{out}$  denote the interface of  $N$ . We further require that neither the initial nor a final marking marks any interface place  $p \in P_{io}$ .  $\lrcorner$

The behavior of an open net (i.e. enabledness and firing of transitions) is defined using the standard (place/transition) Petri net semantics (see [3], for instance), that is, a transition is enabled if each place of its preset holds a token. An enabled transition  $t$  can fire in a marking  $m$  by consuming tokens from the preset places and producing tokens for the postset places, yielding a marking  $m'$ . In order to assign an intuitively consistent meaning to *final* markings, we restrict our approach to such open nets where a marking in  $\Omega$  does not enable any transition.

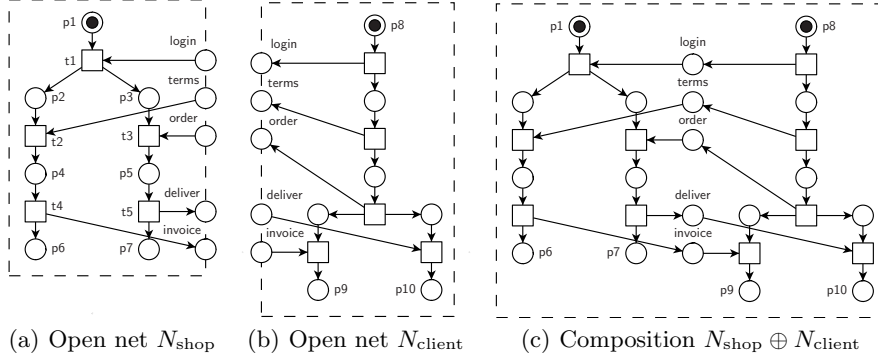
As an example, Fig. 1(a) shows an open net model of an online shop.

Interaction of open nets is represented by their *composition*. Two open nets  $N_1$  and  $N_2$  are *composable* if they only share interface places and the input places of  $N_1$  are exactly the output places of  $N_2$  and vice versa (i.e.  $P_{in_1} = P_{out_2}$  and  $P_{in_2} = P_{out_1}$ ). For two markings  $m_1$  of  $N_1$  and  $m_2$  of  $N_2$ , their *composition*  $m_1 \oplus m_2$  is defined by  $(m_1 \oplus m_2)(p) = m_1(p) + m_2(p)$ . From now on, if two open nets  $N_1$  and  $N_2$  are composed, we implicitly assume they are composable.

**Definition 2 (Composition of open nets).**

The composition of (composable) open nets  $N_1$  and  $N_2$  is the open net  $N = N_1 \oplus N_2$  defined as follows:

- $P = P_1 \cup P_2$ ,



**Fig. 1.** (a) An open net  $N_{\text{shop}}$  modeling an online shop. In the initial marking [p1], it waits for a **login** from a client. After the client logged in, the shop concurrently waits for an **order** which it then will **deliver** and it waits for a confirmation of the **terms** of payment and sends an **invoice** afterwards. Finally, the shop reaches the single final marking [p6, p7]. (b)-(c) An open net  $N_{\text{client}}$  modeling a client of the shop with its final marking [p9, p10] and the composition of shop and client.

- $P_{in} = P_{out} = \emptyset$ ,
- $T = T_1 \cup T_2$ ,
- $F = F_1 \cup F_2$ ,
- $m_0 = m_{0_1} \oplus m_{0_2}$ , and
- $\Omega = \{m_1 \oplus m_2 \mid m_1 \in \Omega_1, m_2 \in \Omega_2\}$ . ┘

A marking  $m$  of an open net  $N$  is a *deadlock* in  $N$  iff  $m$  is no final marking of  $N$  and  $m$  does not enable any transition of  $N$ . This definition of a deadlock differs from the standard definition in literature as we discriminate between terminating (final) states and non-terminating states (i.e. a deadlocks). Deadlock-freedom is a fundamental correctness criterion for cooperating services. In contrast, an open net representing a service in isolation usually has deadlocks. As an example, each of the open nets in Fig. 1(a) and Fig. 1(b) on its own has at least one deadlock, whereas the open net in Fig. 1(c) is deadlock-free.

**Definition 3 (Strategy).**

An open net  $M$  is a (open net) strategy for an open net  $N$  if their composition is deadlock-free.  $\text{Strat}(N)$  denotes the set of all strategies for  $N$ . ┘

If  $\text{Strat}(N) \neq \emptyset$ , then  $N$  is called *controllable*, otherwise  $N$  is *uncontrollable*. Uncontrollable services are fundamentally ill-designed.

Note that according to Definition 3, the strategy notion is symmetric, that is,  $M$  is a strategy for  $N$  iff  $N$  is a strategy for  $M$ . In Sect. 3 we will show how to decide controllability of a given service  $N$  by synthesizing a strategy  $M$ , thus fixing one side of this symmetry. If  $N$  is uncontrollable, then the synthesis produces an “empty strategy”.

Obviously, the client  $N_{\text{client}}$  in Fig. 1(b) is a strategy for the shop  $N_{\text{shop}}$  in Fig. 1(a) (and vice versa). Hence,  $N_{\text{shop}}$  is controllable (and so is  $N_{\text{client}}$ ).

The set  $\text{Strat}(N)$  is of particular importance as it gives a semantics of an open net  $N$  in terms of  $N$ 's deadlock-free interacting environments. In Sections 4–6, we introduce several substitutability notions which all are based on comparing the corresponding sets of strategies.

## 2.2 Service Automata

Service automata [9] form the basis of operating guidelines and are used for representing the behavior of open nets. We will firstly introduce service automata and then present a back and forth translation between open nets and service automata, that uses the set  $\mathcal{MC}$  as interconnection. Service automata are closely related to the reachability graph of the inner of open nets. Thereby, the *inner* of an open net  $N$  is the open net  $\text{inner}(N)$  where all interface places of  $N$  as well as all their adjacent arcs are removed.

### Definition 4 (Service automaton).

A service automaton is an automaton  $A = [Q, I_{\text{in}}, I_{\text{out}}, \delta, q_0, \Omega]$  that consists of

- a set  $Q$  of states,
- two disjoint sets  $I_{\text{in}} \subseteq \mathcal{MC}$  of input channels and  $I_{\text{out}} \subseteq \mathcal{MC}$  of output channels, with  $I_{\text{io}} = I_{\text{in}} \cup I_{\text{out}}$  is the interface of  $A$ ,
- a nondeterministic transition relation  $\delta \subseteq Q \times (I_{\text{io}} \cup \{\tau\}) \times Q$ ,
- a distinguished initial state  $q_0 \in Q$ , and
- a set  $\Omega \subseteq Q$  of final states, such that  $q \in \Omega$  and  $(q, x, q') \in \delta$  implies  $x \in I_{\text{in}}$ . ┘

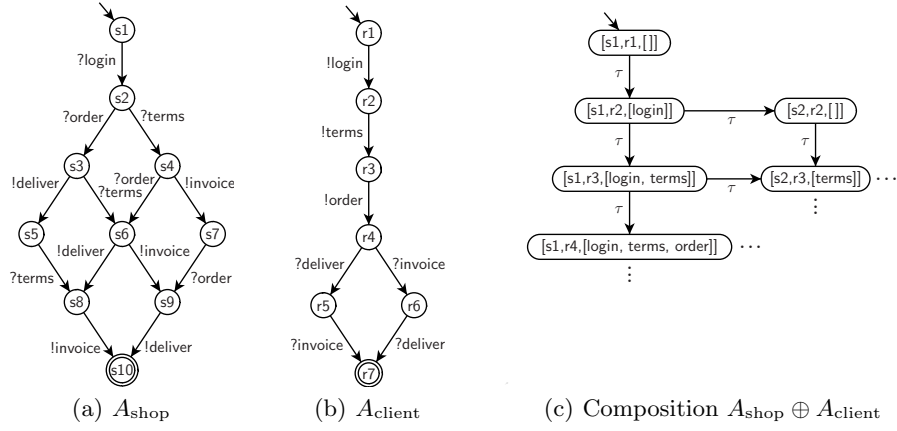
For a transition  $(q, x, q') \in \delta$ ,  $x$  is called the *label* of  $(q, x, q')$ . An  $x$ -labeled transition is a *sending transition* if  $x \in I_{\text{out}}$ , a *receiving transition* if  $x \in I_{\text{in}}$ , and an *internal transition* if  $x = \tau$ . To emphasize the direction of an interface channel  $x \in I_{\text{io}}$  in the graphical representation of a service automaton, we add an exclamation mark,  $!x$ , if  $x \in I_{\text{out}}$ , or a question mark,  $?x$ , if  $x \in I_{\text{in}}$ .

Figure 2 shows three service automata which correspond to the three open nets of Fig. 1.

In the following, we lift notions defined for open nets to service automata.

Two service automata are *composable* if they have disjoint sets of states and the input channels of one automaton are the output channels of the other automaton and vice versa. From now on, we assume all composed service automata are composable.

The *composition*  $A \oplus B$  of composable service automata  $A$  and  $B$  introduces an *internal message bag* (i.e. a multiset) of currently pending messages that were sent by one automaton, but not yet received by the other one. That way, a prior  $x$ -labeled sending transition of  $A$  is represented in  $A \oplus B$  by an internal (i.e.  $\tau$ -labeled) transition that adds one  $x$  element to the message bag  $M$ . Correspondingly, a prior transition receiving an  $x$  is represented by a now internal



**Fig. 2.** Three service automata of the online shop, its client, and their composition.

transition removing an  $x$  from the message bag. Prior internal transitions remain as internal transitions in  $A \oplus B$ . This is formalized in the following definition.

**Definition 5 (Composition of service automata).**

For (composable) service automata  $A$  and  $B$ , their composition is defined as the service automaton  $A \oplus B = [Q, I_{in}, I_{out}, \delta, q_0, \Omega]$  defined as follows:

- $Q = Q_A \times Q_B \times \text{bags}(\mathcal{MC})$ ,
- $I_{in} = I_{out} = \emptyset$ ,
- $\delta \subseteq Q \times \{\tau\} \times Q$ ,
- $q_0 = [q_{0A}, q_{0B}, \emptyset]$ ,
- $\Omega = \Omega_A \times \Omega_B \times \{\emptyset\}$ ,

such that the transition relation  $\delta$  contains the elements

- $([q_A, q_B, M], \tau, [q'_A, q_B, M])$  iff  $(q_A, \tau, q'_A) \in \delta_A$ ,
- $([q_A, q_B, M], \tau, [q_A, q'_B, M])$  iff  $(q_B, \tau, q'_B) \in \delta_B$ ,
- $([q_A, q_B, M], \tau, [q'_A, q_B, M - [x]])$  iff  $(q_A, x, q'_A) \in \delta_A, x \in I_{inA}, M(x) > 0$ ,
- $([q_A, q_B, M], \tau, [q'_A, q_B, M + [x]])$  iff  $(q_A, x, q'_A) \in \delta_A, x \in I_{outA}$ ,
- $([q_A, q_B, M], \tau, [q_A, q'_B, M - [x]])$  iff  $(q_B, x, q'_B) \in \delta_B, x \in I_{inB}, M(x) > 0$ ,
- $([q_A, q_B, M], \tau, [q_A, q'_B, M + [x]])$  iff  $(q_B, x, q'_B) \in \delta_B, x \in I_{outB}$ .  $\lrcorner$

In the rest of this paper, we will only consider the connected part of the service automaton  $A \oplus B$  which contains the initial state (i.e. only states which are  $\delta$ -reachable from  $q_0$ ).

A state  $q$  is a *deadlock* in  $A$  if  $q \notin \Omega$  and at most receiving transitions leave  $q$ . Hence, a service automaton cannot leave a deadlock by its own.

**Definition 6 (Strategy).**

A service automaton  $A$  is a strategy (service automaton) for a service automaton  $B$  if their composition is free of deadlocks.  $\lrcorner$

In analogy to open nets, let  $Strat(A)$  denote the set of all strategies for a service automaton  $A$ .  $A$  is *controllable* iff  $Strat(A) \neq \emptyset$ .

### 2.3 Translating Open Nets into Service Automata and Back

In [10] we have shown that it is possible to transform each open net  $N'$  into a (normalized) open net  $N$  where each transition is connected to at most one interface place while preserving its set of strategies, i.e.  $Strat(N') = Strat(N)$ . Therefore we can, without loss of generality, assume such (normalized) open nets for the transformation into service automata. Let the *inner* of such a (normalized) open net  $N$  be the open net  $inner(N)$  where all interface places of  $N$  as well as all their adjacent arcs are removed.

Then, the service automaton  $A(N)$  of an open net  $N$  is basically the reachability graph of  $inner(N)$ : the states of  $A(N)$  are the reachable markings of  $inner(N)$  and a transition  $t$  of  $inner(N)$  that was connected to an interface place  $p$  in  $N$  becomes a  $p$ -labeled transition of  $A(N)$ . The set  $\mathcal{MC}$  is used as a common name space between the net and its corresponding service automaton, as both the interface places of  $N$  and the interface of  $A(N)$  are subsets of  $\mathcal{MC}$ . If  $t$  was not connected to any interface place in  $N$ , then it becomes a  $\tau$ -labeled (i.e. internal) transition of  $A(N)$ . It is easy to see that the service automata of Fig. 2 can be derived from the open nets of Fig. 1 using this transformation.

Additionally, it is easily possible to translate a service automaton  $A$  into an open net  $N_A$  by constructing a Petri net state machine out of  $A$  or by applying the theory of regions [11], for instance.

Open nets are well-suited to *model* services as they have a more implicit and compact model and are thus more understandable for service designers than service automata. Service automata, in contrast, adequately model service behavior and thus are well-suited to *analyze* services. The value of the back and forth translation is that we can change arbitrarily between these two formalisms without losing information w.r.t.  $Strat$ . Hence, it is sufficient to develop our analysis techniques for service automata, but to model on the Petri net level.

## 3 Operating Guidelines

Operating guidelines were first introduced in [9] and generalized in [10]. Basically, an *operating guideline*  $OG_A$  of a service automaton  $A$  is a special service automaton  $B$  where each state  $q$  of  $B$  is annotated with a Boolean formula  $\phi(q)$ . Such a *Boolean annotated service automaton (BSA)*  $B^\phi$  can be used to characterize a set of service automata [9, 10]. Therefore, we define a matching relation between a service automaton  $B'$  and a Boolean annotated service automaton  $B^\phi$ .  $B^\phi$  characterizes  $B'$  iff  $B'$  *matches* with  $B^\phi$ . An operating guideline  $OG_A$  of a service automaton  $A$  is a special *BSA* where  $B'$  matches with  $OG_A$  iff  $B'$  is a strategy for  $A$ .

A *literal* of our Boolean formulae is a channel in  $\mathcal{MC}$  or one of the special literals  $\tau$  and *final* (representing an internal transition and a final state, respectively). Let, for the rest of this paper,  $\mathcal{MC}^+$  denote the set  $\mathcal{MC} \cup \{final, \tau\}$ . As

Boolean connectors, we only use  $\vee$  (Boolean *or*) and  $\wedge$  (Boolean *and*). Let  $\mathcal{BF}$  be the set of all such Boolean formulae over  $\mathcal{MC}^+$ . As usual, we fix the truth values *true* and *false*. A (Boolean) *assignment* is a mapping  $\beta : \mathcal{MC}^+ \rightarrow \{\text{true}, \text{false}\}$  assigning to each literal a truth value. Furthermore, an assignment  $\beta$  *satisfies* a Boolean formula  $\phi \in \mathcal{BF}$  ( $\beta \models \phi$ ) if  $\phi$  evaluates to *true* using standard propositional logic semantics.

The restriction of *BSAs* to deterministic structures as well as negation-free formulae eases the decision procedures in the upcoming sections while providing all sufficient information needed for operating guidelines.

**Definition 7 (Boolean annotated service automaton, *BSA*).**

A Boolean annotated service automaton (*BSA*)  $B^\phi = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$  consists of

- a deterministic *service automaton*  $B = [Q, I_{in}, I_{out}, \delta, q_0, \Omega]$  and
- a Boolean annotation function  $\phi : Q \rightarrow \mathcal{BF}$ .

Thereby, a *service automaton* is deterministic if it has no internal transitions and each state has at most one  $x$ -labeled outgoing transition. ┘

For matching a *service automaton*  $A$  with a *BSA*  $B^\phi$ , the present outgoing transitions of a state  $q$  of  $A$  constitute an assignment for  $\phi(q)$ :

**Definition 8 (Assignment).**

An assignment of a *service automaton*  $A$  assigns to each state  $q$  of  $A$  a Boolean assignment  $\beta_A(q) : \mathcal{MC}^+ \rightarrow \{\text{true}, \text{false}\}$  defined as follows:

$$\beta_A(q)(x) = \begin{cases} \text{true}, & \text{if } x \neq \text{final and there is a state } q' \text{ with } (q, x, q') \in \delta_A, \\ \text{true}, & \text{if } x = \text{final and } q \in \Omega_A, \\ \text{false}, & \text{otherwise.} \end{cases} \quad \text{┘}$$

A *BSA* is used to characterize a *set* of *service automata*. Let therefore be the matching of a *service automaton* with a *BSA* defined as follows:

**Definition 9 (Matching).**

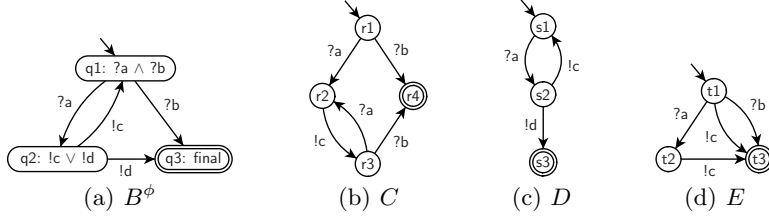
Let  $A$  be a *service automaton* and  $B^\phi$  be a *BSA*.  $A$  matches with  $B^\phi$  if there is a weak simulation relation  $\varrho \subseteq Q_A \times Q_B$  such that for each  $(q_A, q_B) \in \varrho$ :  $\beta_A(q_A) \models \phi(q_B)$ .

Let  $\text{Match}(B^\phi)$  denote the set of all *service automata* that match with  $B^\phi$ . ┘

The weak simulation relation [12]  $\varrho$  together with possible  $\tau$  literals in  $\phi$  allow the deterministic  $B^\phi$  for characterizing deterministic as well as nondeterministic *service automata* [10]. Figure 3(a) shows an example *BSA*. Figures 3(b)– 3(d) demonstrate the matching.

**Definition 10 (Operating guideline, *OG*).**

A Boolean annotated *service automaton*  $OG_A = B^\phi$  is an operating guideline (*OG*) of a *service automaton*  $A$  iff  $\text{Match}(OG_A) = \text{Strat}(A)$ . ┘

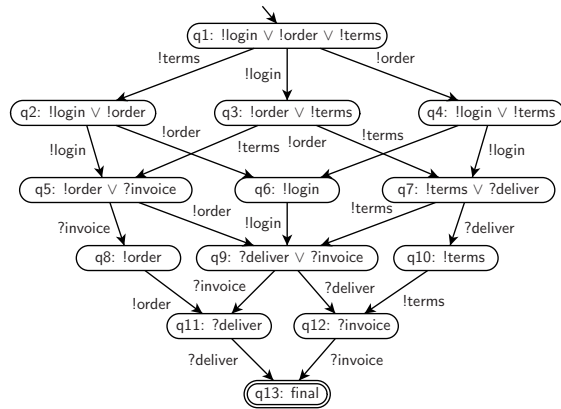


**Fig. 3.** (a) A *BSA*  $B^\phi$ . The annotation  $\phi(q)$  is depicted inside the state  $q$ . (b)–(d) Three service automata  $C$ ,  $D$ , and  $E$ .  $C$  matches with  $B^\phi$ : for instance, the assignment  $\beta_C(r1)$  assigns *true* to the literals  $?a$  and  $?b$  (because both transitions leave the state  $r1$ ), satisfying the annotation  $?a \wedge ?b$ . However,  $D$  and  $E$  do not match with  $B^\phi$ : state  $s1$  does not satisfy the annotation of state  $q1$ ; and the  $!c$ -labeled transition leaving state  $t1$  causes  $B$  not simulating  $E$ .

According to this definition, an operating guideline is a special *BSA*. For uncontrollable service automata  $A$  (i.e.  $Strat(A) = \emptyset$ ) we fix an *OG* that consists of a single state that is annotated with *false*, assuring that *no* service automaton matches with this *OG*.

Figure 4 depicts an operating guideline of our online shop example of Fig. 2(a). Applying Definition 9 we conclude that the client of Fig. 2(b) matches with this *OG*.

In [10] we have presented an algorithm to compute an operating guideline of a service where the inner of the service (cp. Sect. 2.2) has finitely many reachable states. For services without this restriction, we were able to show that controllability is undecidable [13]. The *OG* construction algorithm computes a special strategy. Therefore it starts with an overapproximation of compatible behavior of any strategy and then removes deadlock-causing states iteratively. Finally, the annotations are derived from information collected during the computation. If



**Fig. 4.** An operating guideline of the online shop of Fig. 2(a). To characterize also non-deterministic strategies, each Boolean annotation  $\phi(q)$  is implicitly extended to  $\phi(q) \vee \tau$  and thus evaluated to *true* if the matched service automaton has an outgoing  $\tau$ -transition in the corresponding state (cp. Definition 8).

the service is uncontrollable, the algorithm eventually removes all states. The algorithm is implemented in our tool FIONA [7].<sup>3</sup>

## 4 Accordance

In this section, we define our first substitutability notion, *accordance*. A service  $A'$  accords with a service  $A$  if  $A'$  cooperates with at least the environments that  $A$  cooperates with. That is, if the composition of  $A$  and an environment  $B$  is deadlock-free, then deadlock-freedom is preserved if  $A$  is substituted by  $A'$ .

### 4.1 A Notion of Accordance

Given a service automaton  $A$ , it might be necessary to change or add some functionality of  $A$  by substituting it by a new version  $A'$ . With accordance, we demand that this substitution must not affect any client of  $A$ : every current client of  $A$  has to be supported by  $A'$  as well. Because we assume that  $A$  does not know each client that uses  $A$ ,  $A'$  must support each *potential* clients of  $A$ , i.e. all elements in  $Strat(A)$ . An application for accordance is the upgrade of a web shop which should not affect any client. This motivates the following notion of accordance between service automata  $A$  and  $A'$ . To this end  $A$  and  $A'$  must be *interface equivalent* (i.e.  $I_{inA} = I_{inA'}$  and  $I_{outA} = I_{outA'}$ ).

**Definition 11 (Accordance).**

*Let  $A$  and  $A'$  be interface equivalent service automata.  $A'$  substitutes  $A$  under accordance (short:  $A'$  accords with  $A$ ) iff  $Strat(A) \subseteq Strat(A')$ .  $\lrcorner$*

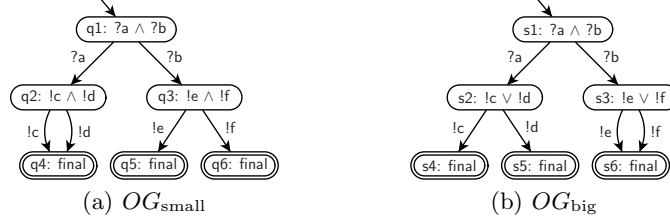
Accordance guarantees that every strategy for  $A$  is a strategy for  $A'$  as well. In other words, if  $A'$  accords with  $A$ , then every client of  $A$  is also a client of  $A'$ . In addition, accordance allows for new clients of  $A'$ . Thus, accordance seems to be the right notion to achieve the goal mentioned above.

The notion of accordance has been first introduced in [14, 15]. However, the decision procedure for accordance was limited to acyclic finite state services there. In this paper, we extend this procedure to cyclic finite state services.

### 4.2 Deciding Accordance

In order to decide accordance of  $A$  and  $A'$ , we need to compare  $Strat(A)$  and  $Strat(A')$ . The problem is that the set  $Strat$  may correspond to a large (in fact infinite) set of service automata. With the operating guidelines of  $A$  and  $A'$  we have, however, a finite representation of  $Strat(A)$  and  $Strat(A')$ . In the following, we show how accordance can be decided by using operating guidelines. To this end we define a refinement relation  $\sqsubseteq$  for operating guidelines. Informally,  $OG_A \sqsubseteq OG_{A'}$ , that is,  $OG_{A'}$  refines  $OG_A$  iff there is a simulation relation between the states of  $OG_A$  and  $OG_{A'}$  such that the annotations in  $OG_A$  imply the annotations in  $OG_{A'}$ .

<sup>3</sup> FIONA is available at <http://www.service-technology.org/fiona>



**Fig. 5.** Two operating guidelines with  $OG_{\text{small}} \sqsubseteq OG_{\text{big}}$ . For instance,  $(q2, s2) \in \xi$  with  $\phi(q2) \Rightarrow \phi(s2)$ , and  $(q4, s4) \in \xi$  and  $(q4, s5) \in \xi$  with  $\phi(q4) \Rightarrow \phi(s4)$  and  $\phi(q4) \Rightarrow \phi(s5)$ .

**Definition 12 (Relation  $\sqsubseteq$  of OGs).**

Let  $A$  and  $A'$  be interface equivalent service automata and let  $OG_A = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$  and  $OG_{A'} = [Q', I'_{in}, I'_{out}, \delta', q'_0, \Omega', \phi']$  be the corresponding operating guidelines. Then,  $OG_A \sqsubseteq OG_{A'}$  iff there is a simulation relation  $\xi \subseteq Q \times Q'$  such that for all  $(q, q') \in \xi$ , the formula  $\phi(q) \Rightarrow \phi'(q')$  is a tautology.  $\lrcorner$

The relation  $\sqsubseteq$  is a preorder, that is, it is reflexive and transitive. By help of the next theorem we show that  $OG_{A'}$  refines  $OG_A$  iff  $A'$  accords with  $A$  and thus it can be used to decide accordance of  $A$  and  $A'$ . An example is depicted in Fig. 5.

**Theorem 1 (Checking accordance).**

Let  $A$  and  $A'$  be two service automata and let  $OG_A$  and  $OG_{A'}$  be the corresponding operating guidelines. Then,  $OG_A \sqsubseteq OG_{A'}$  iff  $Strat(A) \subseteq Strat(A')$ .  $\lrcorner$

For the proof of this theorem we rely on a fact about operating guidelines as constructed in [10]. As we cannot repeat the whole approach of [10], we only include the following proposition and prove Theorem 1.

**Proposition 1.**

For every operating guideline  $OG_A = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$  (of some controllable service automaton  $A$ ) and all  $q \in Q$ , the formula  $\phi(q)$

- (1) uses only literals  $x$  where there is some  $q' \in Q$  with  $(q, x, q') \in \delta$ , and
- (2) is satisfied for the assignment assigning true to all literals in  $\phi(q)$ .  $\lrcorner$

**Proof (of Theorem 1).**

Let  $OG_A = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$  and  $OG_{A'} = [Q', I'_{in}, I'_{out}, \delta', q'_0, \Omega', \phi']$  be the OGs of service automata  $A$  and  $A'$ , respectively.

( $\Rightarrow$ ): Let  $OG_A \sqsubseteq OG_{A'}$  and let  $B$  be an arbitrary strategy service automaton for  $A$ . We show that  $B$  is a strategy for  $A'$ , too.

Because of  $B \in Strat(A)$  there is by Definition 9 a simulation relation  $\varrho \subseteq Q_B \times Q$  between the states of  $B$  and  $OG_A$  and, because of  $OG_A \sqsubseteq OG_{A'}$  there is by Definition 12 a simulation relation  $\xi \subseteq Q \times Q'$  between the states of  $OG_A$  and  $OG_{A'}$ . Define a relation  $\varrho' \subseteq Q_B \times Q'$  between the states of  $B$  and  $OG_{A'}$  such that  $(q_B, q') \in \varrho'$  iff there is a state  $q$  of  $OG_A$  such that  $(q_B, q) \in \varrho$  and  $(q, q') \in \xi$ . We show that  $\varrho'$  is a simulation relation.

Let  $q_B \in Q_B$  and suppose there is a transition  $(q_B, x, q'_B) \in \delta_B$ . From  $\varrho$  being a simulation relation follows that there is a  $q \in Q$  with  $(q_B, q) \in \varrho$ ,  $(q, x, q_1) \in \delta$ , and  $(q'_B, q_1) \in \varrho$ . By  $\xi$  being a simulation relation there is a  $q'$  such that  $(q, q') \in \xi$ ,  $(q', x, q'_1) \in \delta'$ , and  $(q_1, q'_1) \in \xi$ . According to the definition of  $\varrho'$  we have  $(q_B, q')$  and  $(q'_B, q'_1)$  in  $\varrho'$ , and hence we conclude that  $\varrho'$  is a simulation relation between the states of  $B$  and  $OG_{A'}$ .

Next we show for all  $(q_B, q') \in \varrho'$  that  $q_B$  satisfies  $\phi'(q')$ . As  $B$  matches with  $OG_A$ , for all states  $q_B$  with  $(q_B, q) \in \varrho$ ,  $q_B$  satisfies  $\phi(q)$  (for the assignment described in Definition 9). By  $OG_A \sqsubseteq OG_{A'}$  we know  $\phi(q) \Rightarrow \phi'(q')$  for all  $(q, q') \in \xi$ . Hence,  $q_B$  satisfies  $\phi'(q')$  for all  $(q_B, q') \in \varrho'$ , too.

Thus,  $B$  is a strategy for  $A'$  and, hence, we conclude  $\text{Strat}(A) \subseteq \text{Strat}(A')$ .

( $\Leftarrow$ ): Let  $\text{Strat}(A) \subseteq \text{Strat}(A')$ . We show that  $OG_A \sqsubseteq OG_{A'}$ .

Consider the underlying service automaton  $B = [Q, I_{in}, I_{out}, \delta, q_0, \Omega]$  of  $OG_A$ . By construction, the transition systems of  $B$  and  $OG_A$  are isomorphic and hence there is a weak simulation relation  $\varrho$  between the states of  $B$  and  $OG_A$ . From Proposition 1 we conclude that  $B$  satisfies the annotations of  $OG_A$ . Hence,  $B$  is a strategy for  $A$  and thus, by  $\text{Strat}(A) \subseteq \text{Strat}(A')$ , a strategy for  $A'$ . Being a strategy for  $A'$ , there is a simulation relation  $\varrho' \subseteq Q_B \times Q'$  between the states of  $B$  and  $OG_{A'}$ . As  $B$  and  $OG_A$  are isomorphic,  $\varrho'$  constitutes a simulation relation  $\xi \subseteq Q \times Q'$  between the states of  $OG_A$  and  $OG_{A'}$ , too.

Next we show for all  $(q, q') \in \xi$ ,  $\phi(q) \Rightarrow \phi'(q')$ . Let  $q \in Q$  and let  $\beta$  be an arbitrary assignment to literals occurring in  $\phi(q)$  where  $\phi(q)$  is true. Remove from  $B$  all transitions  $(q_1, x, q_2)$  where  $\beta(x)$  is false. By Definition 9, the resulting service automaton is still a strategy for  $A$  and thus a strategy for  $A'$ . Using Definition 9 again, we can see that  $\beta$  satisfies  $\phi'(q')$  as well. Thus,  $\phi(q) \Rightarrow \phi'(q')$  is a tautology for all  $(q, q') \in \xi$ . Hence we conclude  $OG_A \sqsubseteq OG_{A'}$ .

The value of this theorem is that accordance can be checked independently of the environments that  $A$  cooperates with and only  $A$  and  $A'$  have to be known to decide accordance. In order to design a service automaton  $A'$  that accords with  $A$ , a designer can either try and check the resulting service or he derives  $A'$  from  $A$  by applying accordance-preserving transformation rules [14].

For an implementation of the criteria in Theorem. 1, finding the relation  $\xi$  is the crucial task. As both  $OG_A$  and  $OG_{A'}$  are deterministic, this task actually amounts to a depth-first search through  $OG_{A'}$  which is mimicked in  $OG_A$ . The time and space required for finding  $\xi$  is thus linear in the number of states and edges of  $OG_{A'}$ . This size, in turn, is equal to the number of states and edges of a particular strategy for  $A$  [16]. The accordance check based on Theorem. 1 has been implemented in our tool FIONA.

## 5 Restriction

In this section, we introduce another substitutability notion, *restriction*. Restriction is — as accordance — used to compare the sets of environments of two service automata  $A$  and  $A'$ . The goal of restriction is to preserve at least a *fixed*

subset of the environments of  $A$  by  $A'$  (instead of *all* environments of  $A$  as in the accordance setting).

### 5.1 A Notion of Restriction

Given a service automaton  $A$ , we may want to preserve at least a fixed subset  $\mathcal{S} \subseteq \text{Strat}(A)$  of its strategies when substituting  $A$  by a service automaton  $A'$ . This means, every service automaton  $S \in \mathcal{S}$  is a strategy for both  $A$  and  $A'$ . In contrast to the notion of accordance, here we assume that  $A$  *has* knowledge of its environments. To motivate the need of such a substitutability notion, consider again an upgrade of a web shop. Applications for restriction include: the upgraded shop only supports behavior which is used by major clients and all other clients have to adjust their services; the shop restricts itself to its core competencies and rejects all unprofitable strategies; the shop restricts its behavior to certain scenarios such as payment by credit card, for instance. These considerations lead us to the following definition of restriction.

**Definition 13 (Restriction).**

*Let  $A$  and  $A'$  be interface equivalent service automata and let  $\mathcal{S} = \{S_1, \dots, S_n\} \subseteq \text{Strat}(A)$ . Then,  $A'$  substitutes  $A$  under restriction to  $\mathcal{S}$  (short:  $A'$  preserves  $\mathcal{S}$ ) iff  $\mathcal{S} \subseteq \text{Strat}(A')$ .  $\lrcorner$*

According to this definition, at least every service automaton in  $\mathcal{S}$  is a strategy for  $A'$ , meaning, the substitution preserves at least strategies  $\mathcal{S}$ . Hence, restriction seems to be the right notion to achieve the above mentioned goal.

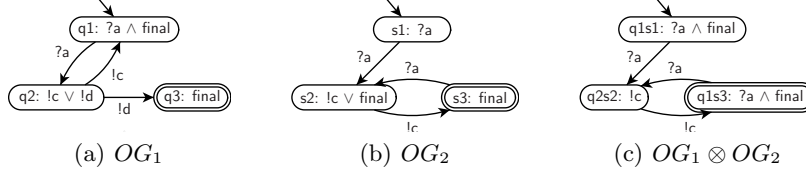
### 5.2 Deciding Restriction

The aim of this section is to introduce a decision procedure whether substituting a service automaton  $A$  by a service automaton  $A'$  preserves a set  $\mathcal{S} \subseteq \text{Strat}(A)$  of strategies. Therefore, we have to check that every service automaton  $S \in \mathcal{S}$  is a strategy for  $A'$ . This decision procedure becomes particularly complex if the set  $\mathcal{S}$  contains many service automata and we want to check several  $A'$ . Therefore, we consider the following alternative: since the notion of a strategy is symmetric, it is equivalent to check whether  $A'$  is a strategy for all  $S \in \mathcal{S}$ . In other words,  $A' \in \bigcap_{S \in \mathcal{S}} \text{Strat}(S)$  must hold.

We will show that the intersection  $\bigcap_{S \in \mathcal{S}} \text{Strat}(S)$  of sets of strategies can be represented by the *product of the operating guidelines* of all service automata  $S \in \mathcal{S}$ . We start by defining the product  $OG_A \otimes OG_B$  of two operating guidelines  $OG_A$  and  $OG_B$  of service automata  $A$  and  $B$  as an operating guideline which characterizes exactly the intersection  $\text{Strat}(A) \cap \text{Strat}(B)$ . To this end  $OG_A$  and  $OG_B$  must be interface equivalent, that is, their underlying automata must be interface equivalent.

**Definition 14 (Product of OGs).**

*Let  $OG_A = C_1^{\phi_1}$  and  $OG_B = C_2^{\phi_2}$  be two (interface equivalent) operating guidelines with  $C_i = [Q_i, I_{in_i}, I_{out_i}, \delta_i, q_{0_i}, \Omega_i]$ ,  $i \in \{1, 2\}$ . Define  $\varrho \subseteq Q_1 \times Q_2$  inductively as follows: let  $(q_{0_1}, q_{0_2}) \in \varrho$ . If  $(q_1, q_2) \in \varrho$ ,  $(q_1, x, q'_1) \in \delta_1$ , and  $(q_2, x, q'_2) \in \delta_2$ , then  $(q'_1, q'_2) \in \varrho$ .*



**Fig. 6.** Two operating guidelines and their product.

Then, the product  $OG_A \otimes OG_B = [Q, I_{in}, I_{out}, \delta, q_0, \Omega, \phi]$  of  $OG_A$  and  $OG_B$  is defined by

- $Q = \{(q_1, q_2) \mid (q_1, q_2) \in \varrho\}$ ,
- $I_{in} = I_{in_1} = I_{in_2}$ , and  $I_{out} = I_{out_1} = I_{out_2}$ ,
- $((q_1, q_2), x, (q'_1, q'_2)) \in \delta$  iff  $(q_1, x, q'_1) \in \delta_1$  and  $(q_2, x, q'_2) \in \delta_2$ ,
- $q_0 = (q_{0_1}, q_{0_2})$ ,
- $\Omega = \{(q_1, q_2) \in Q \mid q_1 \in \Omega_1, q_2 \in \Omega_2\}$ , and
- $\phi((q_1, q_2)) = \phi_1(q_1) \wedge \phi_2(q_2)$ , for all  $(q_1, q_2) \in Q$ . ┘

In a way, the product of operating guidelines is defined analogously to the product of finite automata [17]. Figure 6 shows two interface equivalent operating guidelines and their product (with  $\varrho = \{(q_1, s_1), (q_2, s_2), (q_1, s_3)\}$ ). We assume !d is an output channel of  $OG_2$ , even though there is no !d-labeled transition in  $OG_2$ . Note that the annotations of  $OG_1 \otimes OG_2$  are equivalently minimized: the annotation of state  $\phi(q_1s_1) = (?a \wedge final) \wedge ?a$  is equivalent to  $?a \wedge final$ . Furthermore, the annotation  $\phi(q_2s_2) = (!c \vee !d) \wedge (!c \vee final)$  can be minimized to  $(!c) \wedge (!c \vee final)$  (which is equivalent to !c) because there is no outgoing !d-labeled transition at state  $q_2s_2$ .

Next, we prove that the product of two operating guidelines characterizes indeed the intersection of the strategies represented by these operating guidelines.

**Theorem 2 (Product OG characterizes intersection).**

Let  $OG_A$  and  $OG_B$  be operating guidelines and  $OG_{\otimes} = OG_A \otimes OG_B$  be their product. Then,  $Match(OG_{\otimes}) = Match(OG_A) \cap Match(OG_B)$ . ┘

**Proof.**

Let  $OG_A = [Q_A, I_{inA}, I_{outA}, \delta_A, q_{0A}, \Omega_A, \phi_A]$ ,  $OG_B = [Q_B, I_{inB}, I_{outB}, \delta_B, q_{0B}, \Omega_B, \phi_B]$  and  $OG_{\otimes} = OG_A \otimes OG_B = [Q, I_{in}, I_{out}, \delta, (q_{0A}, q_{0B}), \Omega, \phi]$ .

( $\Rightarrow$ ) Let  $C \in Match(OG_{\otimes})$  and let  $\varrho_{\otimes}$  be a simulation relation between  $C$  and  $OG_{\otimes}$ . We will show that  $C \in Match(OG_A)$  and  $C \in Match(OG_B)$ , too.

Let  $(q_C, (q_A, q_B)) \in \varrho_{\otimes}$  be arbitrary. As  $\varrho_{\otimes}$  is simulation relation there is a sequence  $\sigma$  such that  $q_C$  is reached from  $q_{0C}$  via  $\sigma$  in  $C$  and  $(q_A, q_B)$  is reached from  $(q_{0A}, q_{0B})$  via  $\sigma$  in  $OG_{\otimes}$ . By construction of  $OG_{\otimes}$ ,  $q_{0A}$  and  $q_{0B}$  are reached via  $\sigma$  in  $OG_A$  and  $OG_B$ , too. By Definition 9 again, we have  $(q_C, q_A) \in \varrho_A$  and  $(q_C, q_B) \in \varrho_B$ . Let there be an  $x$ -transition leaving  $q_C$ . From  $C \in Match(OG_{\otimes})$  and Definition 9 (i.e. the weak simulation relation), we can conclude that there is an  $x$ -transition leaving  $(q_A, q_B)$ , too. By the construction of  $\delta$  in Definition 14,

there is an  $x$ -transition leaving  $q_A$  and one leaving  $q_B$ . Hence,  $q_A$  of  $OG_A$  and  $q_B$  of  $OG_B$  simulate  $q_C$ , too.

Furthermore, we conclude from  $C \in \text{Match}(OG_{\otimes})$  and Definition 9 that the assignment  $\beta_C(q_C)$  satisfies  $\phi((q_A, q_B))$ . Hence, by the construction of  $\phi$  in Definition 14,  $\beta_C(q_C)$  also satisfies  $\phi_A(q_A)$  and  $\phi_B(q_B)$ . Consequently,  $C$  matches with  $OG_A$  and  $OG_B$  and therefore  $C \in \text{Match}(OG_A) \cap \text{Match}(OG_B)$ .

( $\Leftarrow$ ) Let  $C \in \text{Match}(OG_A)$  and  $C \in \text{Match}(OG_B)$ . We show that  $C \in \text{Match}(OG_{\otimes})$ .

By  $C \in \text{Match}(OG_A)$  and Definition 9 there is a simulation relation  $\varrho_A$  between  $C$  and  $OG_A$ . Let  $(q_C, q_A) \in \varrho_A$  be arbitrary. As  $\varrho_A$  is simulation relation there is a sequence  $\sigma$  such that  $q_C$  is reached from  $q_{0C}$  via  $\sigma$  in  $C$  and  $q_A$  is reached via  $\sigma$  in  $OG_A$ . By  $C \in \text{Match}(OG_B)$  and Definition 9, there is a simulation relation  $\varrho_B$  and a state  $q_B$  such that  $(q_C, q_B) \in \varrho_B$  and  $q_B$  is reached via  $\sigma$  in  $OG_B$ . By the construction of  $\delta$  in Definition 14,  $(q_A, q_B)$  is reachable in  $OG_{\otimes}$  via  $\sigma$ , too. The rest of the proof follows the same argumentation as above.

The product  $\otimes$  of operating guidelines is commutative and associative, that is, for operating guidelines  $OG_A, OG_B, OG_C$  holds  $OG_A \otimes OG_B = OG_B \otimes OG_A$  and  $(OG_A \otimes OG_B) \otimes OG_C = OG_A \otimes (OG_B \otimes OG_C)$ . So we conclude that the product operating guideline represents exactly the intersection of all strategy sets for service automata in  $\mathcal{S}$ :

**Corollary 1.**

Let  $\mathcal{S} = \{S_1, \dots, S_n\}$  be a set of interface equivalent service automata and let  $OG_{S_i}$  be the operating guideline of  $S_i$ , for all  $1 \leq i \leq n$ . Let  $OG_{\otimes}$  denote the product of all  $OG_{S_i}$ . Then,  $\text{Match}(OG_{\otimes}) = \bigcap_{S \in \mathcal{S}} \text{Strat}(S)$ .  $\lrcorner$

With the help of the above corollary we can prove a theorem which shows that substituting  $A$  by  $A'$  preserves  $\mathcal{S}$  iff  $A'$  is a strategy represented by the product operating guideline  $OG_{\otimes}$ .

**Theorem 3 (Restriction check with product OGs).**

Let  $A$  and  $A'$  be service automata and let  $\mathcal{S} = \{S_1, \dots, S_n\} \subseteq \text{Strat}(A)$ . Let  $OG_{S_i}$ ,  $1 \leq i \leq n$ , be the operating guideline of  $S_i$  and let  $OG_{\otimes}$  denote the product of all  $OG_{S_i}$ . Then,  $A'$  preserves  $\mathcal{S}$  iff  $A' \in \text{Match}(OG_{\otimes})$ .  $\lrcorner$

**Proof.**

We will show that  $\text{Match}(OG_{\otimes})$  characterizes all service automata  $A'$  that can substitute  $A$  while preserving  $\mathcal{S}$ . We have:

$$\begin{aligned} \text{Match}(OG_{\otimes}) &= \bigcap_{S \in \mathcal{S}} \text{Strat}(S) \text{ (by Corollary 1)} \\ &= \{A' \mid \text{for all } S \in \mathcal{S} : A' \in \text{Strat}(S)\} \\ &= \{A' \mid \text{for all } S \in \mathcal{S} : S \in \text{Strat}(A')\} \text{ (because strategy is symmetric)} \\ &= \{A' \mid A' \text{ preserves } \mathcal{S}\} \text{ (by Definition 13)} \end{aligned}$$

Consequently, the theorem holds.

In order to decide whether substituting  $A$  by  $A'$  preserves  $\mathcal{S} \subseteq \text{Strat}(A)$ , we have to construct the operating guideline for each  $S \in \mathcal{S}$  and then calculate the product of these operating guidelines. Time and space complexity for calculating the product of two operating guidelines is proportional to the product of their states. Therefore, this complexity effort only pays off if we check several  $A'$ . The restriction check based on Theorem 3 has been implemented in our tool FIONA.

Intuitively, the fewer strategies shall be preserved by the substitution (i.e. the smaller  $\mathcal{S}$  is), the more service automata  $A'$  exist that may substitute  $A$  (i.e. the bigger is  $\text{Match}(OG_{\otimes})$ ). Because accordance requires all strategies for  $A$  to be preserved by  $A'$ , but restriction requires only a subset of  $A$ 's strategies to be preserved by  $A'$ , there are less services  $A'$  that accord with  $A$  than services  $A'$  that satisfy restriction. Note that for  $\mathcal{S} = \text{Strat}(A)$  restriction coincides with accordance.

As an advantage, the notion of deprecation provides with  $OG_{\otimes}$  an abstract representation of all substituting service automata  $A'$ . So, with the help of  $OG_{\otimes}$ , a service automaton  $A'$  can be *derived* from  $OG_{\otimes}$ , such that  $A'$  substitutes  $A$  while preserving  $\mathcal{S}$ . For instance, the underlying service automaton of  $OG_{\otimes}$  itself serves as a valid  $A'$ . Theorem 3 states that every element in  $\text{Match}(OG_{\otimes})$  can be used to substitute  $A$ . Together with the transformation of a service automaton  $A'$  into a corresponding open net  $N_{A'}$ , we are immediately able to derive substituting open nets as well.

In case of accordance, in contrast, we can only *check* an already given  $A'$  whether it accords to  $A$  or not. To support the construction of according services, we developed accordance-preserving transformation rules to derive from  $A$  a service automaton  $A'$  that accords with  $A$  by construction [14].

## 6 Derived Substitutability Notions

In this section, we introduce two more substitutability notions. Both notions can be derived from the notions of accordance and restriction.

### 6.1 Equivalence

The first substitutability notion we derive is a notion of *equivalence* for service automata. This can be achieved easily by restricting the notion of accordance. Two service automata are equivalent iff they have the same set of strategies.

**Definition 15 (Equivalence).**

*Let  $A$  and  $A'$  be interface equivalent service automata. Then,  $A'$  equivalently substitutes  $A$  (short:  $A'$  and  $A$  are equivalent) iff  $\text{Strat}(A) = \text{Strat}(A')$ .  $\lrcorner$*

Obviously, in order to check equivalence of two service automata, we can check equivalence of their respective operating guidelines. Since equivalence means accordance in both directions, we apply Theorem 1 in both directions.

**Corollary 2 (Checking equivalence with OGs).**

*Two operating guidelines  $OG_A$  and  $OG_B$  are equivalent iff  $OG_A \sqsubseteq OG_B$  and  $OG_A \sqsupseteq OG_B$ .  $\lrcorner$*

## 6.2 Constraints

For many substitutability scenarios the three notions of substitutability we have introduced so far are well-suited. However, there are other scenarios in practice that require less restrictive notions. Accordance demands to preserve all strategies for a given service, even those which are practically infeasible: consider that a service  $A$  has to interact with *two* other services,  $B$  and  $C$ . Assume that  $A$  sends a request to either service  $B$  or  $C$  and concurrently expects an acknowledgement from the respective service. There is a strategy  $S$  for  $A$  such that  $S$  receives the request which  $A$  has sent to  $B$  and acknowledges on behalf of  $C$ . This is, in fact, a valid strategy, but practically impossible if  $B$  and  $C$  do not communicate with each other. This problem arises in the decentralized setting [18]. Such strategies need not to be preserved when substituting  $A$  by  $A'$ .

As another example, if we want to restrict the set of strategies to profitable strategies or to enforce or exclude certain scenarios (e.g. payment by credit card), then restriction is too inflexible, because we would have to identify all infeasible strategies.

These examples motivate the introduction of a notion of a *constraint*. Such a constraint can be seen as a behavioral pattern or communication scenario. We will show how to restrict a set of strategies to those strategies that *enforce* or *exclude* certain behavioral patterns. In [19] such constraints have been introduced to characterize all strategies for a service that conform to a constraint. This approach is used to filter *service registries* for services that fit respective strategies and for validating services by checking whether there exist strategies that access certain features. In contrast to [19], we are interested in services that preserve all strategies that conform to a constraint.

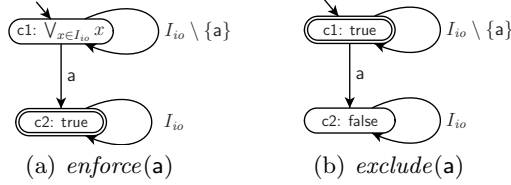
In the following, we define the notion of a constraint *BSA*  $C^\psi$ . Intuitively,  $C^\psi$  is a *BSA* that constrains send and receive actions of an operating guideline  $OG_A$ . Here, to constrain means to enforce or to exclude the respective actions of  $OG_A$ .

### Definition 16 (Constraint *BSA*).

*Let  $A$  and  $C$  be two interface equivalent service automata. Let  $OG_A$  be an operating guideline of  $A$  and let  $\psi$  be an annotation to  $C$ . Then,  $C^\psi$  is a constraint *BSA* for  $OG_A$ .* ┘

Intuitively,  $OG_A$  represents the set of strategies for  $A$  and the constraint *BSA*  $C^\psi$  describes the behavior we want to allow or disallow in the restricted subset of strategies. Therefore, their product characterizes all strategies for  $A$  that conform to  $C^\psi$ . Figure 7 depicts generic constraint automata for enforcing or excluding a communication action  $a$ .

Given a product  $OG_A \otimes C^\psi$ , each service automaton  $A'$  where  $OG_{A'}$  characterizes exactly these strategies is a well-suited candidate for substituting  $A$ . This yields a more fine-grained notion of substitutability under restriction which is covered by the following definition.



**Fig. 7.** Generic constraint automata to enforce or exclude a communication action  $a$ .

**Definition 17 (Constraint-conforming substitution).**

Let  $A, A'$  be service automata and  $OG_A, OG_{A'}$  be the corresponding operating guidelines. Let  $C^\psi$  be a constraint BSA for  $OG_A$ . Then, the substitution of  $A$  by  $A'$  conforms to  $C^\psi$  iff  $Match(OG_{A'}) = Match(OG_A \otimes C^\psi)$ .  $\lrcorner$

In analogy to our accordance check based on Theorem 1, this definition provides a *verification* method only. That is, a *given*  $A'$  can be checked whether it is a valid constraint-conforming substitution of  $A$  or not. The notion of constraints and the substitutability check based on Definition 17 has been implemented in our tool FIONA.

## 7 Related Work

*Compatibility* (e.g. [20]) is the ability of two services  $N$  and  $M$  to interact properly. The literature distinguishes between *structural* and *behavioral* compatibility. Structural compatibility demands that the interfaces of both services match whereas behavioral compatibility demands the control flow of the composition  $N \oplus M$  to fulfill some property such as termination. In our model, we demand strong structural compatibility; that is, for each input channel of  $M$ , there must be an output channel of  $N$  and vice versa. This is due to our definition of composition that requires a closed system. On the level of message exchange, however, we only require weak structural compatibility, meaning, every message that can be sent by  $M$  can be received by  $N$  and vice versa. Behavioral compatibility, in contrast, ensures that in our model the composition is deadlock-free. Stronger criteria such as absence of deadlocks and livelocks are possible, too. Hence the notion of a strategy we used in this paper is a synonym for compatibility:  $M$  and  $N$  are (structural and behavioral) compatible if and only if they are strategies.

A service implementation  $N'$  is *consistent* (e.g. [20]) with a service specification  $N$  if every compatible service  $M$  for  $N$  is compatible to  $N'$ . Thereby, consistency is a synonym for *conformance*, a notion used in process theory (e.g. [21–23]). Conformance is a refinement relation between services (i.e. a preorder). In this paper, we have introduced two consistency notions: accordance which guarantees (the compatibility notion) deadlock-freedom and restriction for which we have extended compatibility to support specific behavior besides deadlock-freedom. Remember that restriction required the preservation of a certain set of strategies.

Substitutability is a collection of consistency notions (together with their corresponding compatibility notions) and each consistency notion corresponds to a specific substitutability scenario. The problem of deciding substitutability

is strongly related to the *compositionality problem* (e.g. [24]) which refers to constructing a system from components while preserving certain properties of the whole system such as the absence of deadlocks and livelocks.

Various substitutability notions can be found in literature. However, most of them lack of an asynchronous communication model as it is necessary in the context of SOC or efficient decision algorithms; or they are too restrictive.

Vogler presents in [24] a livelock and deadlock preserving equivalence between Petri nets with interfaces. However, there is no direct implication in either direction between the equivalence of Vogler and accordance.

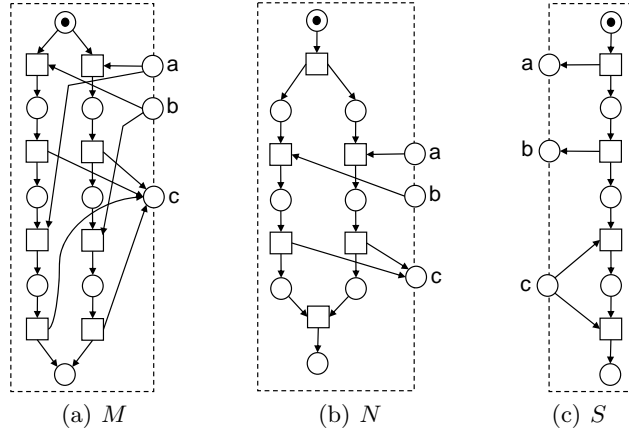
For workflow nets (WFNs) [5] the notion of *inheritance* [25, 26] is used to relate two WFNs that can be substituted. Inheritance bases on branching bisimulation. As a difference, the inheritance approach assumes a synchronous communication model (i.e. transition fusion). In [14], our notion of accordance has been proven to be more liberal than the notion of projection inheritance, that is, projection inheritance implies accordance.

Martens [27] presents a refinement notion for workflow modules. Workflow modules — like open nets — are a Petri net formalism to model (web) services. However, workflow modules are subject to several syntactic restrictions (similar to workflow nets) and therefore less general than open nets. To decide refinement of acyclic workflow modules, Martens introduces a data structure called communication graph and a simulation relation on these graphs. A communication graph in some sense represents the communication behavior of a service and can be compared to a reduced version of our underlying automaton  $B$  of an operating guideline  $OG = B^\phi$  without any annotations. Due to the limitations of workflow modules and the loss of information by the reductions compared to our OGs, his structural simulation relation on communication graphs is only sufficient for refinement (accordance, in our terms) of services, whereas we were able to prove a criterion that is necessary and sufficient. This is proven in Fig. 8.

Bonchi et al. [28] also model the behavior of services with Petri nets. They propose *saturated bisimulation* as equivalence notion. Saturated bisimulation is, however, too restrictive to allow reordering of sending messages. For example, two nets that differ in the order of sending two messages  $a$  and  $b$  are not saturated bisimilar.

In [29–32] automata models are used to decide substitutability. All these approaches use only synchronous communication whereas we consider asynchronous message passing. Benatallah et al. [31] present four notions of substitutability. In this paper, we cover all of them. Equivalence and subsumption mean in our notion equivalence and accordance. In the third notion, service  $S$  can be substituted by  $S'$  assuming the environment  $E$  is known. In this setting, we would check whether  $S$  is a strategy for  $E$ . Finally, in the fourth notion,  $S$  is substituted by  $S'$  w.r.t. an interaction role  $R$ , that is, the intersection of  $S$  and  $R$  has to behave as  $S'$ . In our notion we would check if  $OG_R \otimes OG_S \sqsubseteq OG_{S'}$ .

Refinement relations similar to our notion of accordance are also used in the setting of service contracts. Several refinement relations have been proposed in literature.



**Fig. 8.** Martens’ refinement notion is only sufficient: According to [27],  $S$  is a strategy for  $N$  and not for  $M$  and thus  $N$  refines  $M$ . However, from  $Strat(M) = Strat(N)$  we derive that  $M$  and  $N$  are equivalent.

Bravetti and Zavattaro [21] model services as CCS-like processes. Their proposed conformance relation is stricter than accordance because besides the absence of deadlocks it also excludes livelocks and infinite runs. To decide conformance, the authors prove “*should testing*” [33] to be a sufficient criterion for their conformance notion. As the main difference, our algorithm to decide accordance, which is based on operating guidelines, is complete in the sense that it is sufficient and necessary and it is implemented in our analysis tool FIONA. In [34], Bravetti and Zavattaro enhance their correctness criterion by ensuring whenever a message can be sent, the other service is ready to receive this message. Systems that behave this way are called strongly compliant.

Castagna et al. [22] introduce a conformance notion that formalizes the absence of deadlocks and livelocks in finite-state systems. In contrast to accordance and other conformance notions, their conformance notion only demands the termination of the environment but not the termination of the process itself. In [35] this conformance notion has been proven equivalent to “*must testing*” [36]. Note that “*must testing*” is a weaker equivalence notion than “*should testing*” as it cannot distinguish between a loop (i.e. a potentially infinite run) and a livelock.

Our accordance notion is also related to *testing* (see [37, 38], for instance). A test for a process  $N$  is a process  $S$  such that  $S$  can always terminate in the composition with  $N$  (as required by the conformance notion of [22]). So it is easy to see that every strategy is a specific test that guarantees the termination of the test *and* the process. Our accordance criterion requires proper termination of both components. As two uncontrollable processes are not necessarily structurally related, accordance seems to be different from most preorders in literature. Thus, there seems to be no obvious relationship between accordance and a known preorder. A detailed comparison of accordance and other preorders is, however, outside the scope of this paper.



**Fig. 9.** Accordance does not imply stuck-free conformance of [23]:  $N'$  accords with  $N$ , but  $N'$  does not conform to  $N$ .

Fournet et al. [23] consider CCS processes of asynchronous message passing software components. They present stuck-free conformance of such processes. Stuck-freeness formalizes as accordance the absence of deadlocks in the system. To check conformance, the model checker Zing [39] is used. Stuck-free conformance requires among others that an implemented process  $N'$  simulates its original process  $N$ . Our approach, in contrast, requires a simulation relation between operating guidelines of  $N$  and  $N'$ ; that is, we do not compare  $N$  and  $N'$ , but their strategies. Figure 9 depicts two processes  $N$  and  $N'$  where  $N'$  accords with  $N$ , but  $N'$  does not conform to  $N$ . The reason is, after having received  $b$ , the net  $N'$  can receive  $c$  and  $N$  does not. Thus,  $N'$  does not simulate  $N$ . Therefore, it seems that our notion of accordance is coarser than stuck-free conformance.

Busi et al. [40] present a notion of conformance between a choreography language based on WS-CDL and an orchestration language based on abstract WS-BPEL. This conformance notion is branching bisimulation. Conformance can be used to check if the implementation (i.e. the orchestrated system) behaves accordingly the conversation rules of the choreography. Like saturated bisimulation in [28], branching bisimulation is also too restrictive to allow reordering of messages.

The *ComFoRT* framework [41] analyzes whether a software component  $S$  implemented in the programming language  $C$  can be substituted by another software component  $S'$ .  $S$  can be substituted by  $S'$  if the following two criteria hold: (i) Every behavior possible in  $S$  must also be a behavior of  $S'$ , and (ii) the new version of the software system must satisfy previously established correctness properties. Behavior inclusion is verified by trace comparison of the software components, which also does not allow the reordering of messages.

Pathak et al. [42] focus on a substitutability notion that preserves certain properties of a service  $S$  to be substituted. The properties are expressed by a  $\mu$ -calculus formula  $\phi$ . Then, a  $\mu$ -calculus formula  $\psi$  is calculated such that all services  $S'$  that satisfy  $\psi$  can substitute  $S$ . Due to the expressiveness of the  $\mu$ -calculus in comparison to our proposed constraints on visible actions of open nets, this approach generalizes of our property-preserving substitution, but it assumes, however, a synchronous communication model.

Finally, the idea of using annotated automata as a representation of a set of automata has been first published in [43].

## 8 Conclusion

We have investigated the problem whether a service  $S$  can be substituted by another service  $S'$ . Based on our formal models of open nets and service automata, we have defined different substitutability notions for services: accordance, restriction (in two variants), and equivalence. That way we can formally support various substitutability scenarios which may occur in practice.

As our substitutability notions compare the infinite sets of all deadlock-free interacting services for  $S$  and  $S'$ , the presented decision algorithms apply the concept of an operating guideline as a finite representation of these sets of services. That way we can decide accordance and equivalence for  $S$  and  $S'$ . In addition, we defined the notion of a product operating guideline to specify the intersection of the services represented by several operating guidelines. Product operating guidelines are well-suited to characterize all deadlock-free interacting services for a fixed set of services and can therefore be used for deciding restriction and the more fine-grained restriction notion of property-preserving substitutability.

We implemented all decision algorithms presented in this paper in our analysis tool FIONA. The main functionality of FIONA is to calculate an operating guideline of a service modeled as an open net. With the help of the compiler BPEL2oWFN we can translate WS-BPEL processes into our formal model open nets. Using FIONA we can decide on the open net model whether these WS-BPEL processes can be substituted according to one of the presented substitutability notions. That way we can apply our results to practical applications.

In ongoing research, we will work on other termination criteria than deadlock-freedom. This includes the absence of livelocks and the absence of infinite runs. We will also continue working out a precise relationship of our accordance notion with preorders known from process algebra.

## References

1. Papazoglou, M.P.: Web Services: Principles and Technology. Pearson - Prentice Hall, Essex (2007)
2. Papazoglou, M.P., Traverso, P., Dustdar, S., Leymann, F., Krämer, B.J.: 05462 Service-Oriented Computing: A Research Roadmap. In Curbera, F., Krämer, B.J., Papazoglou, M.P., eds.: Service Oriented Computing (SOC), 15.-18. November 2005. Volume 05462 of Dagstuhl Seminar Proceedings., Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2006)
3. Reisig, W.: Petri Nets. EATCS Monographs on Theoretical Computer Science edn. Springer-Verlag (1985)
4. Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics* **1**(3) (2005) 35–43
5. Aalst, W.M.P.v.d.: The application of Petri nets to workflow management. *Journal of Circuits, Systems and Computers* **8**(1) (1998) 21–66

6. Kindler, E.: A compositional partial order semantics for Petri net components. In Azéma, P., Balbo, G., eds.: *Application and Theory of Petri Nets 1997*, 18th International Conference, ICATPN '97, Toulouse, France, June 23-27, 1997, Proceedings. Volume 1248 of *Lecture Notes in Computer Science.*, Springer-Verlag (1997) 235–252
7. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing Interacting BPEL Processes. In Dustdar, S., Fiadeiro, J.L., Sheth, A., eds.: *Business Process Management*, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings. Volume 4102 of *Lecture Notes in Computer Science.*, Vienna, Austria, Springer-Verlag (2006) 17–32
8. Alves, A., et al.: *Web Services Business Process Execution Language Version 2.0. Committee Specification*, Organization for the Advancement of Structured Information Standards (OASIS) (2007)
9. Massuthe, P., Schmidt, K.: Operating Guidelines - An Automata-Theoretic Foundation for the Service-Oriented Architecture. In Cai, K.Y., Ohnishi, A., Lau, E.M.F., eds.: *5th International Conference on Quality Software (QSIC 2005)*, Melbourne, Australia, IEEE Computer Society (2005) 452–457
10. Lohmann, N., Massuthe, P., Wolf, K.: Operating Guidelines for Finite-State Services. In Kleijn, J., Yakovlev, A., eds.: *Petri Nets and Other Models of Concurrency - ICATPN 2007*, 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007, Siedlce, Poland, June 25-29, 2007, Proceedings. Volume 4546 of *Lecture Notes in Computer Science.*, Siedlce, Poland, Springer-Verlag (2007) 321–341
11. Badouel, E., Darondeau, P.: Theory of Regions. In Reisig, W., Rozenberg, G., eds.: *Lectures on Petri Nets I: Basic Models*, *Advances in Petri Nets*, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996. Volume 1491 of *Lecture Notes in Computer Science.*, Springer-Verlag (1998) 529–586
12. Milner, R.: *Communication and Concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1989)
13. Massuthe, P., Serebrenik, A., Sidorova, N., Wolf, K.: Can I find a partner? Preprint CS-01-08, Universität Rostock, Rostock, Germany (2008) accepted for *Inf. Process. Lett.*
14. Aalst, W.M.P.v.d., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: From Public Views to Private Views – Correctness-by-Design for Services. In Dumas, M., Heckel, R., eds.: *Web Services and Formal Methods*, Forth International Workshop, WS-FM 2007 Brisbane, Australia, September 28-29, 2007, Proceedings. Volume 4937 of *Lecture Notes in Computer Science.*, Springer-Verlag (2008) 139–153
15. Aalst, W.M.P.v.d., Massuthe, P., Stahl, C., Wolf, K.: *Multiparty Contracts: Agreeing and Implementing Interorganizational Processes*. *Informatik-Berichte 213*, Humboldt-Universität zu Berlin (2007) submitted to a journal.
16. Massuthe, P., Wolf, K.: An Algorithm for Matching Non-deterministic Services with Operating Guidelines. *International Journal of Business Process Integration and Management (IJBPIIM)* **2**(2) (2007) 81–90
17. Hopcroft, J.E., Ullman, J.D.: *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley (1979)
18. Schmidt, K.: Controllability of Open Workflow Nets. In Desel, J., Frank, U., eds.: *Enterprise Modelling and Information Systems Architectures*. Volume P-75 of *Lecture Notes in Informatics (LNI).*, Bonn, Entwicklungsmethoden für Informationssysteme und deren Anwendung (EMISA, RWTH Aachen), Köllen Druck+Verlag GmbH (2005) 236–249

19. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral Constraints for Services. In Alonso, G., Dadam, P., Rosemann, M., eds.: Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings. Volume 4714 of Lecture Notes in Computer Science., Springer-Verlag (2007) 271–287
20. Decker, G., Weske, M.: Behavioral Consistency for B2B Process Integration. In Krogstie, J., Opdahl, A.L., Sindre, G., eds.: Advanced Information Systems Engineering, 19th International Conference, CAiSE 2007, Trondheim, Norway, June 11-15, 2007, Proceedings. Volume 4495 of Lecture Notes in Computer Science., Springer-Verlag (2007) 81–95
21. Bravetti, M., Zavattaro, G.: Contract Based Multi-party Service Composition. In Arbab, F., Sirjani, M., eds.: International Symposium on Fundamentals of Software Engineering, International Symposium, FSEN 2007, Tehran, Iran, April 17-19, 2007, Proceedings. Volume 4767 of Lecture Notes in Computer Science., Springer-Verlag (2007) 207–222
22. Castagna, G., Gesbert, N., Padovani, L.: A Theory of Contracts for Web Services. In Necula, G.C., Wadler, P., eds.: Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2008, San Francisco, California, USA, January 7-12, 2008, New York, NY, USA, ACM (2008) 261–272
23. Fournet, C., Hoare, C.A.R., Rajamani, S.K., Rehof, J.: Stuck-Free Conformance. In Alur, R., Peled, D., eds.: Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings. Volume 3114 of Lecture Notes in Computer Science., Springer-Verlag (2004) 242–254
24. Vogler, W.: Modular Construction and Partial Order Semantics of Petri Nets. Volume 625 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (1992)
25. Aalst, W.M.P.v.d., Basten, T.: Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science* **270**(1-2) (2002) 125–203
26. Basten, T., Aalst, W.M.P.v.d.: Inheritance of Behavior. *Journal of Logic and Algebraic Programming* **47**(2) (2001) 47–145
27. Martens, A.: Analyzing web service based business processes. In Cerioli, M., ed.: Fundamental Approaches to Software Engineering, 8th International Conference, FASE 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings. Volume 3442 of Lecture Notes in Computer Science., Edinburgh, UK, Springer-Verlag (2005) 19–33
28. Bonchi, F., Brogi, A., Corfini, S., Gadducci, F.: A Behavioural Congruence for Web Services. In Arbab, F., Sirjani, M., eds.: International Symposium on Fundamentals of Software Engineering, International Symposium, FSEN 2007, Tehran, Iran, April 17-19, 2007, Proceedings. Volume 4767 of Lecture Notes in Computer Science., Springer-Verlag (2007) 240–256
29. Bordeaux, L., Salaün, G., Berardi, D., Mecella, M.: When are Two Web Services Compatible? In Shan, M., Dayal, U., Hsu, M., eds.: Technologies for E-Services, 5th International Workshop, TES 2004. Volume 3324 of Lecture Notes in Computer Science., Springer-Verlag (2004) 15–28
30. Beyer, D., Chakrabarti, A., Henzinger, T.: Web service interfaces. In Ellis, A., Hagino, T., eds.: Proceedings of the 14th international conference on World Wide Web, WWW 2005, ACM (2005) 148–159

31. Benatallah, B., Casati, F., Toumani, F.: Representing, Analysing and Managing Web Service Protocols. *Data Knowl. Eng.* **58**(3) (2006) 327–357
32. Cerná, I., Vareková, P., Zimmerová, B.: Component Substitutability via Equivalencies of Component-Interaction Automata. In: Proceedings of the International Workshop on Formal Aspects of Component Software (FACS'06), Amsterdam, The Netherlands, Elsevier ENTCS (2007) 39–55
33. Rensink, A., Vogler, W.: Fair testing. *Inf. Comput.* **205**(2) (2007) 125–198
34. Bravetti, M., Zavattaro, G.: A theory for strong service compliance. In: Coordination Models and Languages, 9th International Conference, COORDINATION 2007, Paphos, Cyprus, June 6-8, 2007, Proceedings. Volume 4467 of Lecture Notes in Computer Science., Paphos, Cyprus, Springer-Verlag (2007) 96–112
35. Laneve, C., Padovani, L.: The *must* preorder revisited. In Caires, L., Vasconcelos, V.T., eds.: CONCUR 2007 - Concurrency Theory, 18th International Conference, CONCUR 2007, Lisbon, Portugal, September 3-8, 2007, Proceedings. Volume 4703 of Lecture Notes in Computer Science., Lisbon, Portugal, Springer-Verlag (2007) 212–225
36. Nicola, R.D., Hennessy, M.: Testing equivalences for processes. *Theor. Comput. Sci.* **34** (1984) 83–133
37. Glabbeek, R.J.v.: The Linear Time – Branching Time Spectrum I; The Semantics of Concrete, Sequential Processes. In Bergstra, J., Ponse, A., Smolka, S., eds.: Handbook of Process Algebra. Elsevier, Amsterdam, The Netherlands (2001) 3–99
38. Bruda, S.D.: Preorder relations. In Broy, M., Jonsson, B., Katoen, J.P., Leucker, M., Pretschner, A., eds.: Model-Based Testing of Reactive Systems, Advanced Lectures. Volume 3472 of Lecture Notes in Computer Science., Dagstuhl, Germany, Springer (2004) 117–149
39. Andrews, T., Qadeer, S., Rajamani, S.K., Rehof, J., Xie, Y.: Zing: A Model Checker for Concurrent Software. In: Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings. Volume 3114 of Lecture Notes in Computer Science., Springer-Verlag (2004) 484–487
40. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and Orchestration: A Synergic Approach for System Design. In: Service-Oriented Computing - ICSSOC 2005, Third International Conference, Amsterdam, The Netherlands, December 12-15, 2005, Proceedings. Volume 3826 of Lecture Notes in Computer Science., Amsterdam, The Netherlands, Springer-Verlag (2005) 228–240
41. Sharygina, N., Chaki, S., Clarke, E., Sinha, N.: Dynamic Component Substitutability Analysis. In Fitzgerald, J., Hayes, I., Tarlecki, A., eds.: FM 2005: Formal Methods, International Symposium of Formal Methods Europe, Proceedings. Volume 3582 of Lecture Notes in Computer Science., Springer-Verlag (2005) 512–528
42. Pathak, J., Basu, S., Honavar, V.: On Context-Specific Substitutability of Web Services. In: 2007 IEEE International Conference on Web Services (ICWS 2007), July 9-13, 2007, Salt Lake City, Utah, USA, IEEE Computer Society (2007) 192–199
43. Wombacher, A., Fankhauser, P., Mahleko, B., Neuhold, E.J.: Matchmaking for business processes based on choreographies. *International Journal of Web Service Research* **1**(4) (2004) 14–32