Operating Guidelines for Finite-State Services*

Niels Lohmann[1], Peter Massuthe[1], and Karsten Wolf[2]

[1] Humboldt-Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, 10099 Berlin, Germany
{nlohmann,massuthe}@informatik.hu-berlin.de
[2] Universität Rostock, Institut für Informatik,
18051 Rostock, Germany
karsten.wolf@informatik.uni-rostock.de

Abstract. We study services modeled as *open workflow nets* (oWFN) and describe their behavior as *service automata*. Based on arbitrary finite-state service automata, we introduce the concept of an *operating guideline*, generalizing the work of [1, 2] which was restricted to acyclic services.
An operating guideline gives complete information about how to properly interact (in this paper: deadlock-freely and with limited communication) with an oWFN N. It can be executed, thus forming a properly interacting partner of N, or it can be used to support service discovery.
An operating guideline for N is a particular service automaton S that is enriched with Boolean annotations. S interacts properly with the service automaton *Prov*, representing the behavior of N, and is able to simulate every other service that interacts properly with *Prov*. The attached annotations give complete information about whether or not a simulated service interacts properly with *Prov*, too.

1 Introduction

In real life, we routinely use complicated electronic devices such as digital cameras, alarm clocks, mobile phones, CD players, vending machines, etc. Using such a device involves complex interaction, where information from the user to the device flows via pushing buttons or spinning wheels while information is passed from the device to the user via displays or blinking LEDs.

In some cases, we do not even abstractly know what is going on inside the device. Nevertheless, we are typically able to participate in the interaction. Besides ergonomic design, help from experienced friends, or trial-and-error exploration, it is often the user instructions which help us to figure out what to do at which stage. The typical features of user instructions (at least good ones) are:

- they are shipped with, or pinned to, the device,
- they are operational, that is, a user can execute them step by step,
- they are complete, that is, they cover the full intended functionality of the device,

* Partially funded by the BMBF project "Tools4BPEL".

Niels Lohmann, Peter Massuthe, and Karsten Wolf. **Operating guidelines for finite-state services**. In Jetty Kleijn and Alex Yakovlev, editors, *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, pages 321–341. Springer-Verlag, June 2007.

– they use only terms related to the interface (buttons, displays, etc.) without trying to explain the internal processes.

In the virtual world, services [3] replace the devices of the real world. Still, using a service may require involved interaction with the user (which can be another service, like in service-oriented computing [4]). With the concept of an *operating guideline*, we are going to propose an artifact that, in the virtual world, plays the role of user instructions in the real world. In particular, we will show that it exhibits the characteristics listed above. Moreover, we show that the operating guideline for a service can be automatically computed and be used for automatically checking proper interaction between services.

In contrast, a *public view* of a service (a condensed version of the service itself) has been proposed as another artifact for explaining the interaction with the service [5, 6]. Public views, however, do neither match the second nor the fourth item of the list above.

Our approach is based on the behavior of *open workflow nets* (oWFN) [2]. oWFN are a class of Petri nets which has been proposed for modeling services. oWFN generalize and extend the classical *workflow nets* [7]. The most important extension is an interface for asynchronous message passing. This interface allows us to compose services to larger units. Suitability of oWFN for modeling services has been proven through an implemented translation from the industrial service description language WS-BPEL [8] into oWFN [9, 10]. While there are many partial formalizations for WS-BPEL, the translation to oWFN is feature complete. Other feature complete formalizations are based on *abstract state machines* [11, 12].

We describe the *behavior* of an oWFN with the help of a *service automaton*. A service automaton basically records the internal states of an oWFN. The transitions of the automaton are labeled with information about message passing through the mentioned interface. Service automata form the basis of the proposed *operating guidelines*.

Operating guidelines have so far been introduced for *acyclic services* [1, 2]. In this paper, we extend our previous results and introduce the concept of an operating guideline for an *arbitrary finite-state* service N. The operating guideline of N is a distinguished service automaton S that properly interacts with N, together with *Boolean annotations* at each state of S. The annotations serve as a characterization of *all* services that properly interact with N.

For this paper, we assume that "proper interaction" between services N and N' means deadlock freedom of the system composed of N and N' and limited communication, that is, k-boundedness of all message buffers, for some given k. We are well aware that there are other possibilities for defining "proper interaction". Nevertheless, deadlock freedom and limited communication will certainly be part of any such definition, so this paper can be seen as a step towards a more sophisticated setting.

The rest of the paper is organized as follows. In Sect. 2, we introduce open workflow nets, service automata, and their relation. Sections 3 to 5 are devoted to the construction of an operating guideline and its use. These sections build

entirely upon the concept of service automata. We start by defining, in Sect. 3, a concept that we call *situations*. This concept is fundamental to our approach. It describes the interaction of a given service automaton *Prov* with a partner *Req* from the point of view of *Req* only. With the help of situations, we are able to characterize deadlock freedom of the interaction in a way that is suitable for subsequent considerations. The characterization is translated into Boolean formulas which are used as annotations in the operating guideline later on. The calculation and justification of the canonical partner S mentioned above is subject of Sect. 4. In Sect. 5, finally, we formalize the concept of an operating guideline and show how it can be used for identifying other partners that communicate deadlock-freely with *Prov*. Section 6 discusses issues of an implementation and presents experimental results. Finally, we summarize the results of the paper and sketch our plans for further work.

2 Models for Services

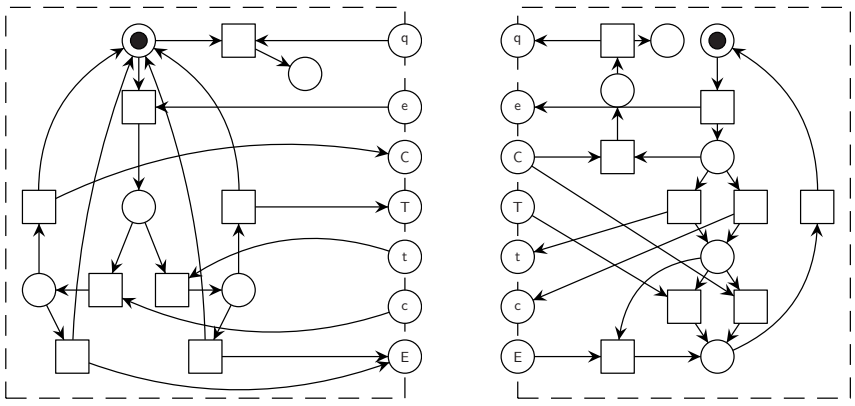
2.1 Open workflow nets (oWFN)

The introduction of open workflow nets [2] was inspired by the view of a service as a workflow plus an interface. Consequently, oWFN extend workflow nets [7] with an interface for asynchronous message passing. oWFN further drop some syntactic restrictions present in workflow nets, for instance the unique start and end places. These restrictions would complicate service composition without sufficient justification by improved analysis possibilities.

Definition 1 (Open workflow net). *An* open workflow net *consists of:*
- *an ordinary place/transition net* $[P, T, F, m_0]$*; together with*
- *two disjoint sets* $P_i, P_o \subseteq P$*, called* input *and* output *places, such that* $F \cap (P_o \times T) = \emptyset$ *and* $F \cap (T \times P_i) = \emptyset$*. We assume* $m_0(p) = 0$ *for* $p \in P_i \cup P_o$*;*
- *a set* Ω *of markings, called* final markings*. For* $m_f \in \Omega$ *and* $p \in P_i \cup P_o$*, we assume* $m_f(p) = 0$*. We further require that a marking in* m_f *does not enable a transition.*

P_i represents channels for incoming messages, P_o channels for outgoing messages. The required restrictions for arcs guarantee that sent messages cannot be "unsent" and received messages cannot be "unreceived". Our set of final markings replaces the single final place of workflow nets. Any marking may be final as long as it does not enable a transition. That is, a service does not perform actions in a final marking. A service may, however, be designed such that it resumes work when it receives a message while residing in a final marking.

A major intention behind services is their composition to larger units. Correspondingly, there is a concept of composition for oWFN. oWFN are composed by merging interface places. This can, in general, be done for arbitrarily many oWFN. For the purpose of this paper, it is sufficient to understand the composition of just two oWFN.



(a) A vending machine oWFN N_V. (b) A partner oWFN of N_V.

Fig. 1. The oWFN N_V (a) models a vending machine. In every iteration, it first expects a coin to be inserted (message e), and a choice for a coffee or a tea to be made (c or t). It returns either the coin (E), modeling a failed check for validity, or the corresponding beverage (C or T). The machine can be shut down by sending q. The oWFN (b) models a potential partner (user) of the vending machine.

Definition 2 (Partner oWFN). *Two oWFN N and N' are partners if $P_i = P'_o$ and $P_o = P'_i$. All other ingredients of N and N' are assumed to be disjoint.*

If two oWFN are partners, they can be composed. Composition consists mainly of merging the interface places.

Definition 3 (Composition of oWFN). *Let N and N' be partner oWFN. The composition $m \oplus m' : P \cup P' \to \mathbb{N}$ of two markings m of N and m' of N' is defined by $(m \oplus m')(p) = m(p)$, if $p \in P$ and $(m \oplus m')(p) = m'(p)$, if $p \in P'$. The* composition of N and N' *is the oWFN $N \oplus N'$ defined as follows:*

- $P_{N \oplus N'} = P \cup P'$, $T_{N \oplus N'} = T \cup T'$, $F_{N \oplus N'} = F \cup F'$, $m_{0_{N \oplus N'}} = m_0 \oplus m'_0$;
- $P_{i_{N \oplus N'}} = P_{o_{N \oplus N'}} = \emptyset$;
- $\Omega_{N \oplus N'} = \{m \oplus m' \mid m \in \Omega, m' \in \Omega'\}$.

The composition of markings is well-defined for partners, as the common places of N and N' do not carry tokens. The composition of partners leads to an oWFN with an empty interface. As an example, Fig. 1 shows two partner oWFN.

Only for oWFN with empty interface it is reasonable to consider their reachability graph (occurrence graph), as an interface suggests some interaction with a (possibly unknown) environment. We rely on the usual concept of reachability graph. For studying a service in isolation, we consider the *inner* of an (uncomposed) oWFN. The inner of N is an oWFN with empty interface, so its reachability graph may be considered.

Definition 4 (Inner). *Let N be an oWFN. The* inner of N*, denoted $inner(N)$, is obtained from N by removing all places in P_i and P_o and their adjacent arcs. Initial and final markings are adjusted accordingly.*

This leads to the following definition of boundedness of arbitrary oWFN.

Definition 5 (Boundedness of oWFN). *An oWFN N is* bounded *if the reachability graph of $inner(N)$ is finite.*

Boundedness, as defined above, concerns the inner of an oWFN. The composition of two bounded oWFN, however, can still be unbounded since tokens may be accumulated in the merged interface places. Thus, we have to introduce an additional concept of boundedness of the interface places.

Definition 6 (Limited communication of oWFN). *Two partner oWFN N and N' have k-*limited communication *(for some $k \in \mathbb{N}$) if $m(p) \leq k$ for all markings m reachable in $N \oplus N'$ and all places $p \in P_i \cup P'_i$.*

If two bounded oWFN N and N' are k-limited partners, then $N \oplus N'$ is bounded, too.

2.2 Service automata

Open workflow net models as introduced so far can be obtained from practical specifications of services. There is, for instance, a feature complete translation from WS-BPEL to oWFN [9, 10].

In this section we introduce *service automata* [1], which serve as the basis of the calculation of operating guidelines. A state of a service automaton is comparable to a marking of the inner of an oWFN. Communication activities are modeled as annotations to the transitions of a service automaton.

Service automata differ from standard I/O-automata [13]. They communicate asynchronously rather than synchronously, and they do not require explicit modeling of the state of the message channels. This approach leads to smaller and thus more readable automata. Other versions of automata models for services were proposed by [14] and [3], for instance. [14] model communication as occurrences of labels with no explicit representation of pending messages, whereas [3] use bounded and unbounded queues to store such messages.

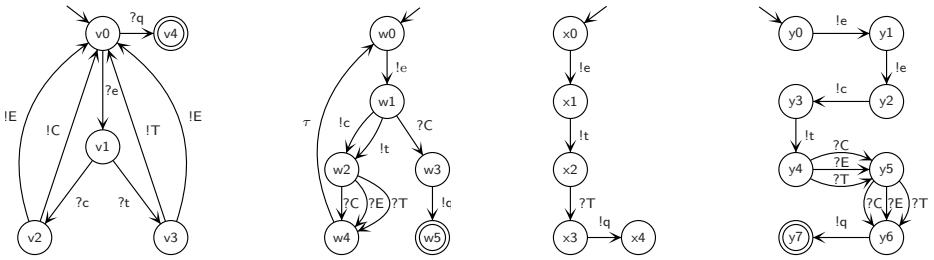
Unlike an oWFN, a single service automaton has no explicit concept of message channels. The channels are taken care of in the definition of composition: a state of a composed service automaton consists of a state of each participating service automaton and a state of the *message bag* of currently pending messages.

We fix a finite set C, the elements of which we call *channels*. They take the role of the interface places in oWFN. We assume $\tau \notin C$ (the symbol τ is reserved for an internal move). With $bags(C)$, we denote the set of all multisets over C, that is, all mappings $m : C \to \mathbb{N}$. A multiset over C models a state of the message bag, that is, it represents, for each channel, the number of pending messages. $[\,]$ denotes the empty multiset ($[\,](x) = 0$ for all x), $[x]$ a singleton

multiset ($[x](x) = 1$, $[x](y) = 0$ for $y \neq x$), $m_1 + m_2$ denotes the sum of two multisets ($(m_1 + m_2)(x) = m_1(x) + m_2(x)$ for all x), and $m_1 - m_2$ the difference ($(m_1 - m_2)(x) = max(m_1(x) - m_2(x), 0)$ for all x). $bags_k(C)$ denotes the set of all those multisets m over C where $m(x) \leq k$ for all x. $bags_k(C)$ is used for modeling the concept of limited communication.

Definition 7 (Service automata). *A* service automaton $A = [Q, I, O, \delta, q_0, F]$ *consists of a set Q of* states*, a set $I \subseteq C$ of* input channels*, a set $O \subseteq C$, $I \cap O = \emptyset$ of* output channels*, a nondeterministic* transition relation $\delta \subseteq Q \times (I \cup O \cup \{\tau\}) \times Q$*, an* initial state $q_0 \in Q$*, and a set of* final states $F \subseteq Q$ *such that $q \in F$ and $[q, x, q'] \in \delta$ implies $x \in I$. A is* finite *if its set of states is finite.*

Throughout this paper, we use the following notations for service automata. With *Prov* (from service *prov*ider), we denote an arbitrary service automaton for which we are going to calculate its operating guideline. With *Req* (from service *req*uester), we denote an arbitrary service automaton in its role as a communication partner of *Prov*. S is used for the particular partner of *Prov* that forms the core of the operating guideline for *Prov*. Service automata without an assigned role are denoted A. We use indices to distinguish the constituents of different service automata. In figures, we represent a channel $x \in I$ with $?x$ and a channel $y \in O$ with $!y$. Figure 2 shows four examples of service automata.



(a) V (b) W (c) X (d) Y

Fig. 2. Examples of service automata. The service automaton V models our vending machine (see Fig. 1(a)). The service automata W, X, and Y model partners of V. Final states are depicted by double circles.

Definition 8 (Partner automata). *Two service automata A and B are* partner *automata if $I_A = O_B$ and $I_B = O_A$.*

As in the case of oWFN, partner automata can be composed.

Definition 9 (Composition of service automata). *For partner automata A and B, their* composition *is defined as the service automaton $A \oplus B = [Q_{A\oplus B}, I_{A\oplus B}, O_{A\oplus B}, \delta_{A\oplus B}, q_{0_{A\oplus B}}, F_{A\oplus B}]$ defined as follows:*
$Q_{A\oplus B} = Q_A \times Q_B \times bags(C)$, $I_{A\oplus B} = O_{A\oplus B} = \emptyset$, $q_{0_{A\oplus B}} = [q_{0_A}, q_{0_B}, [\,]]$, $F_{A\oplus B} = F_A \times F_B \times \{[\,]\}$. *The transition relation $\delta_{A\oplus B}$ contains the elements*

– $[[q_A, q_B, m], \tau, [q'_A, q_B, m]]$ *iff* $[q_A, \tau, q'_A] \in \delta_A$ *(internal move in A),*
– $[[q_A, q_B, m], \tau, [q_A, q'_B, m]]$ *iff* $[q_B, \tau, q'_B] \in \delta_B$ *(internal move in B),*
– $[[q_A, q_B, m], \tau, [q'_A, q_B, m - [x]]]$ *iff* $[q_A, x, q'_A] \in \delta_A$*,* $x \in I_A$*, and* $m(x) > 0$ *(receive by A),*
– $[[q_A, q_B, m], \tau, [q_A, q'_B, m - [x]]]$ *iff* $[q_B, x, q'_B] \in \delta_B$*,* $x \in I_B$*, and* $m(x) > 0$ *(receive by B),*
– $[[q_A, q_B, m], \tau, [q'_A, q_B, m + [x]]]$ *iff* $[q_A, x, q'_A] \in \delta_A$ *and* $x \in O_A$ *(send by A),*
– $[[q_A, q_B, m], \tau, [q_A, q'_B, m + [x]]]$ *iff* $[q_B, x, q'_B] \in \delta_B$ *and* $x \in O_B$ *(send by B),*
and no other elements.

Figure 3 depicts the composition $V \oplus W$ of the services V and W of Fig. 2.

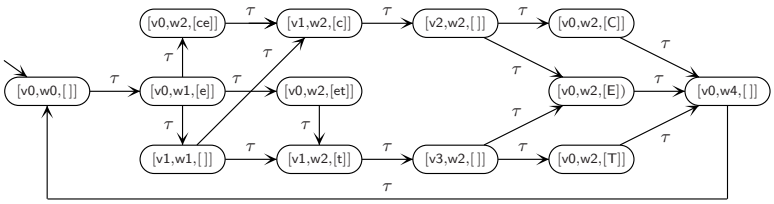


Fig. 3. The composed system $V \oplus W$ of the service automata V and W of Fig. 2. Only states reachable from the initial state are depicted. Note that $V \oplus W$ has no (reachable) final states. Nevertheless, $V \oplus W$ is deadlock-free, which is central in this paper.

Definition 10 (Wait state, deadlock). *For an automaton A, a state q is called a* wait state *iff* $[q, x, q'] \in \delta$ *implies* $x \in I$*, that is, q cannot be left without help from the environment. For a wait state q, let* $wait(q) = \{x \in I \mid \exists q' \in Q : [q, x, q'] \in \delta\}$*. A wait state q is called* deadlock *iff* $q \notin F$ *and* $wait(q) = \emptyset$*.*

A wait state cannot be left without an incoming message. $wait(q)$ is the set of all incoming messages that would help to leave q. A deadlock cannot be left, independently from incoming messages. The definition of service automata requires final states to be wait states which is reasonable.

Examples for wait states in Fig. 2 are $\mathsf{v0}$ with $wait(\mathsf{v0}) = \{\mathsf{e}, \mathsf{q}\}$, $\mathsf{w2}$ with $wait(\mathsf{w2}) = \{\mathsf{C}, \mathsf{E}, \mathsf{T}\}$, or $\mathsf{x4}$ with $wait(\mathsf{x4}) = \emptyset$. An example for a deadlock is the state $[\mathsf{v0}, \mathsf{x2}, [\mathsf{E}]]$ of the (not depicted) composition of the services V and X of Fig. 2 that can be reached from the initial state $[\mathsf{v0}, \mathsf{x0}, [\,]]$ of $V \oplus X$ by executing first the transitions send e and send t of service X, followed by the transitions receive e, receive t, and send E of service V.

For service automata, limited communication can be formalized as follows.

Definition 11 (Limited communication of service automata). *Let A and B be two partner automata and* $A \oplus B$ *their composition. Then, A is called a* k-limited communication partner *of B iff* $Q_{A \oplus B} \subseteq Q_A \times Q_B \times bags_k(C)$*.*

Throughout the paper, we assume k to be given and fixed. The value of k may be chosen either by considerations on the physical message channels, by a static analysis that delivers a "sufficiently high" value, or just randomly. If two finite service automata *Prov* and *Req* are k-limited partners, then $Prov \oplus Req$ is finite as well. In Fig. 2, W and X are 1-limited partners of V. Y is no 1-limited partner since $V \oplus Y$ contains, for instance, the state $[\mathsf{v0}, \mathsf{y2}, [\mathsf{ee}]]$. Y is, however, a 2-limited partner of V.

Every k-limited communication partner is a $(k+1)$-limited communication partner as well. For every value of k, there are services which have a deadlock-free k-limited communication partner but no deadlock-free $(k-1)$-limited communication partner. There are even services which have deadlock-free communication partners but not a k-limited one for any k. As an example, consider the service in Fig. 5(a): A communication partner would have a single (initial and final) state s in which it receives a and loops back to s.

2.3 Translation from oWFN to service automata

While there exist direct translations from WS-BPEL to automata and closely related formalisms [15–17] we propose to generate service automata from oWFN. This way, we can directly inherit the already mentioned feature completeness of the Petri net translations of WS-BPEL [9, 10].

Comparing the behavior of oWFN and service automata, the main difference is the capability of oWFN to send and receive several messages in a single transition occurrence. In order to keep matters simple, we give a direct translation from an oWFN N to a service automaton only for the case that every transition of N is connected to at most one place in $P_i \cup P_o$. In fact, this assumption holds for all oWFN stemming from WS-BPEL specifications as a BPEL activity accesses at most one message channel. On the other hand, an arbitrary oWFN can be transformed in various ways to match the requirement. We sketch one possible transformation in Fig. 4.

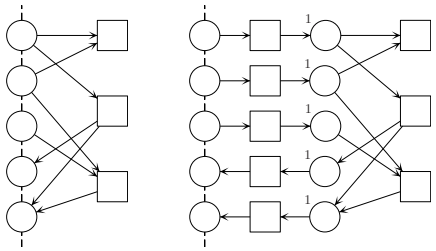


Fig. 4. In the oWFN (a), transitions are connected to several interface places. This can be circumvented by wrapping the interface with an additional internal buffer for each message channel that has capacity one (annotation to the places), cf. (b). This way, the essential behavior of the oWFN as well as finiteness of the state space are preserved.

Given the restriction that a transition of N accesses at most one interface place, the translation from oWFN to service automata is straightforward and formalized through a mapping *oWFNtoService* from oWFN to service automata.

Definition 12 (Mapping oWFN to automata). *Let N be an oWFN where every transition accesses at most one interface place. Then $oWFNtoService(N)$ is the service automaton A with the following constituents:*

- *Q_A is the set of reachable markings of $inner(N)$;*
- *$I_A = P_i$, $O_A = P_o$;*
- *$[m, a, m'] \in \delta_A$ iff there is a transition t of N such that $[m, t, m']$ is a transition in the reachability graph of $inner(N)$ and either there is an interface place p connected to t and $a = p$, or t is not connected to any interface place and $a = \tau$;*
- *q_{0_A} is the initial state of $inner(N)$, F_A is the set of final states of $inner(N)$.*

The translation is justified through the following observation that can be easily verified by induction on the transition relations.

Proposition 1. *For any two oWFN N and N' where every transition is connected to at most one interface place, the reachability graph of $N \oplus N'$ is isomorphic to the graph defined by the states and transitions of $oWFNtoService(N) \oplus oWFNtoService(N')$.*

In the remainder of this article, we study service automata *Prov* and *Req* where *Prov*, *Req*, and *Prov* $\oplus$ *Req* are all finite. This restriction implements the limited communication property introduced earlier. We return to oWFN in Sect. 6 where we discuss our implementation and report experimental results.

3 A Characterization of Deadlocks

In this section, we introduce concepts that help us to understand the coupling between two service automata *Prov* and *Req* from the point of view of *Req*. Therefore, we introduce the concept of *situations* which will allow us to characterize a deadlock in *Prov* $\oplus$ *Req* by considering *Req* only.

Definition 13 (K, Situation). *Let Prov and Req be partners. Then, let $K : Q_{Req} \to 2^{Q_{Prov} \times bags(C)}$ be defined by $K(q_{Req}) = \{[q_{Prov}, m] \mid [q_{Prov}, q_{Req}, m]$ is reachable from the initial state in $Q_{Prov \oplus Req}\}$. The elements of $2^{Q_{Prov} \times bags(C)}$ are called* situations.

A situation comprises all parts of a state of *Prov*$\oplus$*Req* beyond the state of *Req* itself. It can thus be handled independently of *Req*. $K(q_{Req})$ can be interpreted as the *knowledge* that *Req* has about the possible states of *Prov* and the message bag, that is, the situations $[q_{Prov}, m]$ that can occur with q_{Req} in *Prov* $\oplus$ *Req*.

We give some examples for values of K, referring to Fig. 2. We consider W as a partner of V. Then Fig. 3 tells us that $K(\mathsf{w0}) = \{[\mathsf{v0}, [\,]]\}$, $K(\mathsf{w1}) = \{[\mathsf{v0}, [\mathsf{e}]], [\mathsf{v1}, [\,]]\}$, $K(\mathsf{w2}) = \{[\mathsf{v0}, [\mathsf{ce}]], [\mathsf{v0}, [\mathsf{et}]], [\mathsf{v1}, [\mathsf{c}]], [\mathsf{v1}, [\mathsf{t}]], [\mathsf{v2}, [\,]], [\mathsf{v3}, [\,]], [\mathsf{v0}, [\mathsf{C}]], [\mathsf{v0}, [\mathsf{E}]], [\mathsf{v0}, [\mathsf{T}]]\}$, $K(\mathsf{w3}) = \emptyset$, and $K(\mathsf{w4}) = \{[\mathsf{v0}, [\,]]\}$.

Within a set M of situations, we distinguish transient and stable situations. A situation is transient in M if a move of *Prov* in that situation leads to another situation also contained in M. Otherwise it is stable.

Definition 14 (Transient, stable situation). *Let M be a set of situations. $[q_{Prov}, m]$ is* transient *in M iff there is an $[q_{Prov}, x, q'_{Prov}] \in \delta_{Prov}$ such that:*
- *$x = \tau$ and $[q'_{Prov}, m] \in M$, or*
- *$x \in I_{Prov}$, $m(x) > 0$, and $[q'_{Prov}, m - [x]] \in M$, or*
- *$x \in O_{Prov}$ and $[q'_{Prov}, m + [x]] \in M$.*

Otherwise, $[q_{Prov}, m]$ is stable *in M.*

A service cannot leave a stable situation without interaction with the environment. For example, the situation $[\mathsf{v0}, [\mathsf{e}]]$ is transient in the set of situations $K(\mathsf{w1})$ (cf. Fig. 3). In contrast, the situation $[\mathsf{v1}, [\,]]$ is stable in $K(\mathsf{w1})$.

A deadlock in the composed system $Prov \oplus Req$ — seen from the point of view of Req only — now reads as follows.

Lemma 1. *$[q_{Prov}, q_{Req}, m]$ is a deadlock of $Prov \oplus Req$ if and only if all of the following conditions hold:*
- *$q_{Prov} \notin F_{Prov}$, or $q_{Req} \notin F_{Req}$, or $m \neq [\,]$;*
- *q_{Req} is a wait state of Req;*
- *$[q_{Prov}, m]$ is stable in $K(q_{Req})$ and $m(x) = 0$ for all $x \in wait(q_{Req})$.*

Proof. ($\rightarrow$) Let $[q_{Prov}, q_{Req}, m]$ be a deadlock. Then the first item is true by definition of deadlocks. The second item must be true since otherwise Req has a move. $[q_{Prov}, m]$ must be stable since otherwise $Prov$ has a move. For $x \in wait(q_{Req})$, we can conclude $m(x) = 0$ since otherwise Req has a move.

($\leftarrow$) Assume, the three conditions hold. By the first item, the considered state is not a final state of $Prov \oplus Req$. $Prov$ does not have a move since $[q_{Prov}, m]$ is stable. Req does not have a move since internal and send moves are excluded by the second item, and receive moves are excluded by the last item. □

Consider again the example deadlock $[\mathsf{v0}, \mathsf{x2}, [\mathsf{E}]]$ in $V \oplus X$ of the services of Fig. 2 and the three criteria of Lemma 1. Firstly, $[\mathsf{E}] \neq [\,]$. Secondly, $\mathsf{x2}$ is a wait state of X with $wait(\mathsf{x2}) = \{\mathsf{T}\}$. Thirdly, $K(\mathsf{x2}) = \{[\mathsf{v0}, [\mathsf{et}]], [\mathsf{v1}, [\mathsf{t}]], [\mathsf{v3}, [\,]], [\mathsf{v0}, [\mathsf{E}]], [\mathsf{v0}, [\mathsf{T}]]\}$ and $[\mathsf{v0}, [\mathsf{E}]]$ is stable in $K(\mathsf{x2})$ and $[\mathsf{E}](\mathrm{T}) = 0$. Hence, all criteria hold and we can conclude that $[\mathsf{v0}, \mathsf{x2}, [\mathsf{E}]]$ is indeed a deadlock.

For a state $[q_{Prov}, q_{Req}, m]$, the three requirements of Lemma 1 can be easily compiled into Boolean formulas $\phi_1(q_{Prov}, m)$, ϕ_2, and $\phi_3(m)$ which express the absence of deadlocks of the shape $[\cdot, q_{Req}, \cdot]$ in $Prov \oplus Req$. The formulas use the set of propositions $C \cup \{\tau, final\}$ (with $final \notin C$). Propositions in $C \cup \{\tau\}$ represent labels of transitions that leave q_{Req}, whereas proposition *final* represents the fact whether $q_{Req} \in F_{Req}$:

Definition 15 (Annotation, *Req*-assignment). *Let $Prov$ and Req be partners. Then, for each $q_{Req} \in Q_{Req}$, define the* annotation $\phi(q_{Req})$ *of q_{Req} as the Boolean formula over the propositions $C \cup \{\tau, final\}$ as follows.*

$$\phi(q_{Req}) = \bigwedge\nolimits_{[q_{Prov}, m] \text{ stable in } K(q_{Req})} (\phi_1(q_{Prov}, m) \vee \phi_2 \vee \phi_3(m))$$

where

- $\phi_1(q_{Prov}, m) = \begin{cases} final, & \text{if } q_{Prov} \in F_{Prov} \text{ and } m = [\,], \\ false, & \text{otherwise,} \end{cases}$
- $\phi_2 = \tau \vee \bigvee_{x \in O_{Req}} x$,
- $\phi_3(m) = \bigvee_{x \in I_{Req}, m(x) > 0} x$.

The Req-assignment $ass_{Req}(q_{Req}) : C \cup \{\tau, final\} \to \{true, false\}$ *assigns true to a proposition* $x \in C \cup \{\tau\}$ *iff there is a* q'_{Req} *such that* $[q_{Req}, x, q'_{Req}] \in \delta_{Req}$ *and true to the proposition final iff* $q_{Req} \in F_{Req}$.

Since the formula $\phi(q_{Req})$ exactly reflects Lemma 1, we obtain:

Corollary 1. *Prov* $\oplus$ *Req is deadlock-free if and only if, for all* $q_{Req} \in Q_{Req}$, *the value of* $\phi(q_{Req})$ *with the Req-assignment* $ass_{Req}(q_{Req})$ *is true.*

In Fig. 2, the annotation of state $\mathsf{w1}$ of W would be $\tau \vee \mathsf{e} \vee \mathsf{c} \vee \mathsf{t} \vee \mathsf{q}$, due to the single stable situation $[\mathsf{v1}, [\,]] \in K(\mathsf{w1})$. This formula is satisfied by the W-assignment that assigns *true* to both c and t, and *false* to τ, e, and q in state $\mathsf{w1}$. The annotation of the state $\mathsf{w2}$ is $\tau \vee \mathsf{e} \vee \mathsf{c} \vee \mathsf{t} \vee \mathsf{q} \vee (\mathsf{C} \wedge \mathsf{E} \wedge \mathsf{T})$ since $K(\mathsf{w2})$ contains the three stable situations $[\mathsf{v0}, [\mathsf{C}]]$, $[\mathsf{v0}, [\mathsf{E}]]$, and $[\mathsf{v0}, [\mathsf{T}]]$. Since the W-assignment assigns *true* to all of C, E, and T in state $\mathsf{w2}$, it satisfies the annotation. For state $\mathsf{x2}$ of X, the annotation is $\tau \vee \mathsf{e} \vee \mathsf{c} \vee \mathsf{t} \vee \mathsf{q} \vee (\mathsf{T} \wedge \mathsf{E})$. Since the only transition leaving $\mathsf{x2}$ is T, the X-assignment assigns *false* to all propositions except T in state $\mathsf{x2}$, and the annotation yields *false*. This corresponds to the deadlock $[\mathsf{v0}, \mathsf{x2}, [\mathsf{E}]]$ in $V \oplus X$.

4 A Canonical Partner

We are now ready to compute a canonical service automaton, called S, which interacts properly with a given service *Prov*.

For any finite k-limited communication partner of a given (finite) service automaton *Prov*, all reachable situations are actually in $2^{Q_{Prov} \times bags_k(C)}$ which is a finite domain. For sets of situations, define the following operations.

Definition 16 (Closure). *For a set* M *of situations, let the* closure *of* M, *denoted* $cl(M)$, *be inductively defined as follows.*
Base: $M \subseteq cl(M)$;
Step: If $[q_{Prov}, m] \in cl(M)$ *and* $[q_{Prov}, x, q'_{Prov}] \in \delta_{Prov}$, *then*
- $[q'_{Prov}, m] \in cl(M)$, *if* $x = \tau$,
- $[q'_{Prov}, m + [x]] \in cl(M)$, *if* $x \in O_{Prov}$,
- $[q'_{Prov}, m - [x]] \in cl(M)$, *if* $x \in I_{Prov}$ *and* $m(x) > 0$.

It can be easily seen that $cl(M)$ comprises those situations that can be reached from a situation in M without interference from a partner. In Fig. 2 for example, we obtain $cl(\{[\mathsf{v0}, [\mathsf{ce}]]\}) = \{[\mathsf{v0}, [\mathsf{ce}]], [\mathsf{v1}, [\mathsf{c}]], [\mathsf{v2}, [\,]], [\mathsf{v0}, [\mathsf{C}]], [\mathsf{v0}, [\mathsf{E}]]\}$.

Definition 17 (Send-event, receive-event, internal-event). *Let $M \subseteq Q_{Prov} \times bags(C)$. If $x \in O_{Prov}$, then the* send-event x, *$send(M, x)$, is defined as $send(M, x) = \{[q, m + [x]] \mid [q, m] \in M\}$. If $x \in I_{Prov}$, then the* receive-event x, *$receive(M, x)$, is defined as $receive(M, x) = \{[q, m - [x]] \mid [q, m] \in M, m(x) > 0\}$. The* internal-event τ, *$internal(M, \tau)$, is defined as $internal(M, \tau) = M$. As the shape of an event is clear from I_{Prov} and O_{Prov}, we define the* event x, *$event(M, x)$, as $receive(M, x)$ if $x \in I_{Prov}$, $send(M, x)$ if $x \in O_{Prov}$, and $internal(M, x)$ if $x = \tau$.*

A send-event models the effect that a message sent by *Req* has on a set of situations M. A receive-event models the effect that a message received by *Req* has on a set of situations M. Considering the service V of Fig. 2, we get, for example, $receive(\{[\mathsf{v0}, [\mathsf{ce}]], [\mathsf{v1}, [\mathsf{c}]], [\mathsf{v2}, [\,]], [\mathsf{v0}, [\mathsf{C}]], [\mathsf{v0}, [\mathsf{E}]]\}, \mathsf{C}) = \{[\mathsf{v0}, [\,]]\}$ and $send(\{[\mathsf{v0}, [\mathsf{e}]], [\mathsf{v1}, [\,]]\}, \mathsf{c}) = \{[\mathsf{v0}, [\mathsf{ce}]], [\mathsf{v1}, \mathsf{c}]]\}$.

Now, the construction of S (Def. 18) bases on the following considerations. A *state* of S is a *set of situations*. States and transitions are organized such that, for all states q of S, $K(q) = q$, that is, every state of S is equal to the set of situations it can occur with in the composition with *Prov*. The transitions of S can be determined using the operations *event* and *cl*. The construction is restricted to sets in $2^{Q_{Prov} \times bags_k(C)}$. With this restriction, we already implement the property of k-limited communication. Given the desired property $K(q) = q$ and the definition of K, the composed system cannot enter a state violating k-limited communication. The other way round, any reachable state q where $K(q)$ is outside $2^{Q_{Prov} \times bags_k(C)}$ would cause a violation of that property.

Starting with a service automaton S_0 which contains *all* such states and transitions, unfortunately, $S_0 \oplus Prov$ may contain deadlocks. However, these deadlocks can be identified by annotating S_0 and evaluating the annotations according to Def. 15. Removing all states where the annotation evaluates to *false* yields a new structure S_1. Since it is possible that the initial state is among the removed ones, S_1 is not necessarily a service automaton, that is, it is not well-defined. In that case, however, we are able to prove that *Prov* does not have correctly interacting partners at all. By removing states, assignments of remaining states can change their values. Thus, the removal procedure is iterated until either the initial state is removed, or all annotations eventually evaluate to *true*. In the latter case, the remaining service automaton is called S and, by construction of the annotations, constitutes a partner that interacts properly with *Prov*.

Definition 18 (Canonical partner S). *Let Prov be a service automaton and assume a number k to be given. Define inductively a sequence of (not necessarily well-defined) service automata $S_i = [Q_i, I_i, O_i, \delta_i, q_{0i}, F_i]$ as follows.*

Let $Q_0 = 2^{Q_{Prov} \times bags_k(C)}$. Let, for all i, $I_i = O_{Prov}$, $O_i = I_{Prov}$, $q_{0i} = cl(\{[q_{0Prov}, [\,]]\})$, $[q, x, q'] \in \delta_i$ iff $q, q' \in Q_i$ and $q' = cl(event(q, x))$, and $F_i = \{q \in Q_i \mid q$ is wait state of $S_i\}$. Let, for all i, $Q_{i+1} = \{q \mid q \in Q_i, \phi(q)$ evaluates to true with assignment $ass_{S_i}(q)\}$.

Let S be equal to S_i for the smallest i satisfying $S_i = S_{i+1}$.

As the sequence $\{S_i\}_{i=0,1,\ldots}$ is monotonously decreasing, all objects of this definition are uniquely defined. The resulting S is a well-defined service automaton if and only if $q_{0_S} \in Q_S$. In that case, S is in fact a k-limited deadlock-freely interacting partner of *Prov*.

As an example, the partner service S_0 for the service V of Fig. 2 initially consists of 21 (from $q_{0_{S_0}}$ reachable) states from which 9 states are removed during the computation of the canonical partner S for V. The resulting service automaton S can be found as the underlying graph of Fig. 7 in Sect. 5.

With the next few results, we further justify the construction.

Lemma 2. *If* $cl([q_{0Prov}, [\,]]) \not\subseteq Q_{Prov} \times bags_k(C)$*, then Prov does not have* k*-limited communication partners for the number* k *used in the construction.*

Proof. As $cl([q_{0Prov}, [\,]])$ is the set of situations that can be reached from the initial state without interference of any partner *Req*, k-limited communication is immediately violated. □

Lemma 2 states that S_0 is well-defined for all well-designed *Prov*, that is, for all *Prov* such that there exists at least one partner *Req* with $Prov \oplus Req$ is deadlock-free.

The next lemma shows that we actually achieved one of the major goals of the construction.

Lemma 3. *For all* S_i *and all* $q \in Q_i$*: if* q *is* δ_i*-reachable from* q_{0_i}*, then* $K(q) = q$*.*

Proof. By structural induction on δ. By definition of cl, $cl([q_{0Prov}, [\,]])$ is the set of situations that can be reached from the initial state without interference from S_i. If $K(q) = q$, then $cl(event(q, x))$ is by definition of *event* and cl exactly the set of situations that can be reached from situations in q by the event x. Thus, $[q, x, q'] \in \delta_i$ implies $K(q') = q'$. □

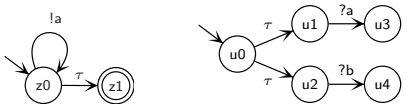
From that lemma we can directly conclude that the service S constitutes a properly interacting partner of *Prov*.

Corollary 2. *If* S *is well-defined, that is,* $q_{0_S} \in Q_S$*, then* $Prov \oplus S$ *is deadlock-free.*

Proof. Follows with Lemma 1 and Def. 15 from the fact that all states of S satisfy their annotations. □

As an example of an ill-designed service, the service Z in Fig. 5(a) would yield an infinite $cl([\mathsf{z0}, [\,]])$ for any partner. Accordingly, there is no well-defined S_0. During the construction of S for service U in Fig. 5(b), the initial state is eventually removed. The initial state $\{[\mathsf{u0}, [\,]], [\mathsf{u1}, [\,]], [\mathsf{u2}, [\,]]\}$ of S_0 for U has two successors. The a-successor $\{[\mathsf{u0}, [\mathsf{a}]], [\mathsf{u1}, [\mathsf{a}]], [\mathsf{u2}, [\mathsf{a}]], [\mathsf{u3}, [\,]]\}$ must be removed since it contains the deadlock $[\mathsf{u2}, [\mathsf{a}]]$, the b-successor must be removed since it contains the deadlock $[\mathsf{u1}, [\mathsf{b}]]$. In the next iteration, the initial state itself must be removed since, without the two successors, it violates its annotation $(\mathsf{a} \wedge \mathsf{b}) \vee \tau$.

For further studying the constructed partner S, we establish a matching relation between states of services and apply this relation to relate states of an arbitrary partner *Req* of *Prov* to states of the canonical partner S.



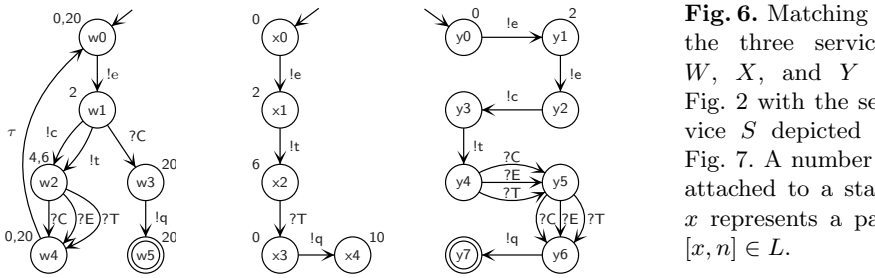
(a) Z (b) U

Fig. 5. The service Z has no k-limited communication partner (for any number k). The service U cannot communicate deadlock-freely with any partner.

Definition 19 (Matching). *Let A and B be service automata and define the relation $L_{A,B} \subseteq Q_A \times Q_B$, the* matching *between A and B, inductively as follows. Let $[q_{0_A}, q_{0_B}] \in L_{A,B}$. If $[q_A, q_B] \in L_{A,B}$, $[q_A, x, q'_A] \in \delta_A$ and $[q_B, x, q'_B] \in \delta_B$, then $[q'_A, q'_B] \in L_{A,B}$.*

The matching between two services A and B is a strong simulation relation where in particular one τ-step of A is related to exactly one τ-step of B.

Examples for matchings are shown in Fig. 6. For example, the state w2 of the service W in Fig. 6(a) is matched with the states 4 and 6 of the service S in Fig. 7.



(a) W (b) X (c) Y

Fig. 6. Matching of the three services W, X, and Y of Fig. 2 with the service S depicted in Fig. 7. A number n attached to a state x represents a pair $[x, n] \in L$.

Lemma 4. *Let S_0 be the starting point of the construction in Def. 18 and Req be a partner of Prov with k-limited communication (for the value of k used in the construction above). For all $q_{Req} \in Q_{Req}$, $K(q_{Req}) = \bigcup_{[q_{Req}, q_{S_0}] \in L_{Req,S_0}} K(q_{S_0})$.*

Proof (Sketch). The inclusion $K(q_{Req}) \supseteq \bigcup_{[q_{Req}, q_{S_0}] \in L_{Req,S_0}} q_{S_0}$ follows from the definition of q_{0S_0}, δ_{S_0}, and the concepts *cl* and *event*. For the reverse inclusion, let $[q_{Prov}, m] \in K_{q_{Req}}$, that is, $[q_{Prov}, q_{Req}, m] \in Prov \oplus Req$. Thus, there is a transition sequence in $Prov \oplus Req$ from the initial state $[q_{0Prov}, q_{0Req}, [\,]]$ to that state. This sequence can be replayed in $Prov \oplus S_0$ by replacing actions of Req with actions of S_0, leading to a state q_{S_0} with $[q_{Prov}, m] \in K(q_{S_0}) = q_{S_0}$. □

Corollary 3. *For each state q_{Req} of Req, its annotation $\phi(q_{Req})$ can be described as $\phi(q_{Req}) \equiv \bigwedge_{q_{S_0}:[q_{Req}, q_{S_0}] \in L_{Req,S_0}} \phi(q_{S_0})$.*

Proof. Since an annotation $\phi(q_{S_0})$ is a conjunction built for every situation in $q_{S_0} = K(q_{S_0})$, the annotation of the union of K-values is the conjunction of the individual formulas. □

For example, the annotation of state $\mathsf{w2}$ of the service in Fig. 2 is $(\mathsf{C} \wedge \mathsf{E} \wedge \mathsf{T}) \vee \mathsf{c} \vee \mathsf{e} \vee \mathsf{t} \vee \tau$ which is equivalent to the conjunction of the annotations $(\mathsf{C} \wedge \mathsf{E}) \vee \tau$ and $(\mathsf{E} \wedge \mathsf{T}) \vee \mathsf{c} \vee \mathsf{e} \vee \mathsf{t} \vee \tau$ of the states $\mathsf{4}$ and $\mathsf{6}$ of the service in Fig. 7.

The next result is the actual justification of the removal process described in Def. 18.

Lemma 5. *If Req is a k-limited communication partner of Prov (for the value of k used in the construction of S_0) such that Prov $\oplus$ Req is deadlock-free, then $q_S \in S$ for all $[q_{Req}, q_S] \in L_{Req,S_0}$.*

Proof. (By contradiction) Let i be the smallest number such that there exist $q_{Req} \in Q_{Req}$ and $q_S \in Q_{S_i} \setminus Q_{S_{i+1}}$ holding $[q_{Req}, q_S] \in L$. That is, q_S is, among the states of S_0 appearing in L_{Req,S_0}, the one that is removed first during the process described in Def. 18.

By the construction of Def. 18, $\phi(q_S)$ evaluates to *false* with the assignment $ass_{S_i}(q_S)$. Thus, there is a $[q_{Prov}, m] \in K(q_S)$ such that $[q_{Prov}, q_S, m]$ is a deadlock in $Prov \oplus S_i$. As a deadlock, it is also a wait state in S_i, so $q_S \in F_{S_i}$.

In S_0, there is, for every $x \in C \cup \{\tau\}$, a transition leaving q_S. If such a transition is not present from q_S in S_i, this means that the corresponding successor state has been removed in an earlier iteration of the process described in Def. 18. Such a transition cannot leave q_{Req} in *Req* since otherwise a successor of *Req* were matched with a state q'_S that has been removed in an earlier iteration than q_S which contradicts the choice of i and q_S. Consequently, for every x with an x-transition leaving q_{Req} in *Req*, there is an x-transition leaving q_S in S_i. This means that, for all $x \in C \cup \{\tau, final\}$, $ass_{S_i}(q_S)(x) \geq ass_{Req}(q_{Req})(x)$. Since $\phi(q_S)$ is monotonous (only built using $\vee$ and $\wedge$), and ϕ_{Req} is a conjunction containing $\phi(q_S)$ (by. Cor. 3), $\phi(q_{Req})$ evaluates to *false* with the assignment $ass_{Req}(q_{Req})$. Consequently, by Cor. 1, $Prov \oplus Req$ has a deadlock. □

Corollary 4. *Prov has a k-limited communication partner Req (for the value of k used in the construction of S_0) such that Prov $\oplus$ Req is deadlock-free if and only if $q_{0S} \in Q_S$ (i. e. the service-automaton S is well-defined).*

Proof. If S is well-defined then, by Cor. 2, at least S is a partner of *Prov* such that $Prov \oplus S$ is deadlock-free. If *Prov* has a partner *Req* such that $Prov \oplus Req$ is deadlock-free, Lemma 5 asserts that L_{Req,S_0} contains only states of S. In particular, since in any case $[q_{0Req}, q_{0S_0}] \in L_{Req,S_0}$, this implies $q_{0S_0} = q_{0S} \in Q_S$. □

If *Prov* does not have partners *Req* such that $Prov \oplus Req$ is deadlock-free, then *Prov* is fundamentally ill-designed. Otherwise, the particular partner S studied above is well-defined. It is the basis for the concept of an operating guideline for *Prov* which is introduced in the next section.

5 Operating Guidelines

If the matching of a service *Req* with S_0 involves states of $Q_{S_0} \setminus Q_S$, Lemma 5 asserts that $Prov \oplus Req$ has deadlocks. In the case that the matching involves only states of Q_S, $Prov \oplus Req$ may or may not have deadlocks. However, by Cor. 1, the existence of deadlocks in $Prov \oplus Req$ can be decided by evaluating the annotations $\phi(q_{Req})$ for the states $q_{Req} \in Q_{Req}$. By Cor. 3, these formulas can be retrieved from the annotations to the states of S. Attaching these formulas explicitly to the states of S, the whole process of matching and constructing the $\phi(q_{Req})$ can be executed without knowing the actual contents of the states of S, that is, without knowing the situations — the topology of S is sufficient. This observation leads us to the concept of an operating guideline for *Prov*.

Definition 20 (Operating guideline). *Let Prov be a service automaton which has at least one properly interacting partner and S be the canonical service automaton of Def. 18. Then any automaton S^* that is isomorphic to S under isomorphism h, together with an annotation Φ with $\Phi(q_{S^*}) = \phi(h(q_{S^*}))$ for all q_{S^*}, is called* operating guideline for *Prov*.

With the step from S to an isomorphic S^*, we just want to emphasize that only the topology of S is relevant in the operating guideline while the internal structure of the states of S, that is, the set of situations, is irrelevant.

Figure 7 shows the operating guideline, that is, the annotated service automaton S, for the service V of Fig. 2.

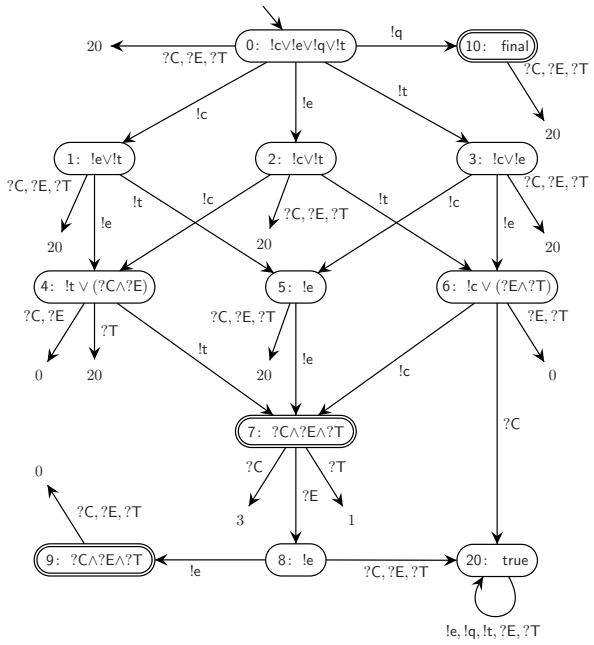


Fig. 7. The operating guideline for the service V in Fig. 2. It is isomorphic to the canonical partner S for V (Def. 18) with the annotations Φ depicted inside the nodes.
Multiple labels to an edge mean multiple edges. Edges pointing to a number mean edges to the node with the corresponding number.

Ignoring the annotations, the operating guideline is (isomorphic to) the partner S for *Prov* that can be executed directly thus satisfying the second requirement stated in the introduction. The annotation at a state q gives additional instructions about whether or not transitions leaving q may be skipped. Therefore, the operating guideline can be used to decide, for an arbitrary service *Req*, whether or not $Prov \oplus Req$ is deadlock-free, as the next result shows.

Theorem 1 (Main theorem of this article). *Let Prov be a finite state service and S^* its operating guideline. Req is a k-limited communication partner of Prov (for the value of k used in the construction of S^*) such that $Prov \oplus Req$ is deadlock-free if and only if the following requirements hold for every $[q_{Req}, q_{S^*}] \in L_{Req,S^*}$:*

(topology) *For every $x \in C \cup \{\tau\}$, if there is an x-transition leaving q_{Req} in Req, then there is an x-transition leaving q_{S^*} in S^*.*

(annotation) *The assignment $ass_{Req}(q_{Req})$ satisfies $\Phi(q_{S^*})$.*

Note that this theorem matches *Req* with S^* (isomorphic to S) while the results in the previous section match *Req* with S_0. Requirement (topology) actually states that L_{Req,S^*} is a simulation relation.

Proof. If $Prov \oplus Req$ is deadlock-free, then Lemma 5 asserts that the matching of *Req* with S (or S^*) coincides with the matching of *Req* with S_0. Thus, requirement (topology) holds. Furthermore, Cor. 3 guarantees that requirement (annotation) is satisfied.

Assume that both requirements hold. By requirement (topology), the matching of *Req* with S (or S^*) coincides with the matching of *Req* with S_0, since the matching with S_0 can lead to states outside S only if there is an x such that an x-transition is present in a state q_{Req} but not in the corresponding state $q_S \in S$. Given that both matchings coincide, Cor. 3 states that $\phi(q_{Req})$ is the conjunction of the $\phi(q_S)$, for the matching states q_S. Then, we can deduce from Cor. 1 and requirement (annotation) that $Prov \oplus Req$ is deadlock-free. □

Consider the service V of Fig. 2 and its partners. In Fig. 6 we can see that W and X satisfy the requirement (topology) while Y does not (Y is not a 1-limited communication partner of V). X violates in state x2 the annotation to the matched state 6, since the *Req*-assignment in state x2 assigns *false* to E and τ. $V \oplus X$ contains the deadlock $[\mathsf{v0}, \mathsf{x2}, [\mathsf{E}]]$. For service W, all annotations are satisfied. $V \oplus W$ is deadlock-free (see Fig. 3).

At this stage, it would be natural to ask for an operating guideline that has the shape of an oWFN rather than an automaton. While the partner S can be transformed to an oWFN using existing approaches called region theory [18], we have no concept of transforming the annotations of the states of S into corresponding annotations of the resulting oWFN. That is why our concept of operating guidelines is presented on the level of service automata.

6 Implementation

All concepts used in this article have been defined constructively. For an actual implementation, it is, however, useful to add some ideas that increase efficiency. First, it is easy to see that, for constructing S, it is not necessary to start with the whole S_0. For the matching, only states that are reachable from the initial state need to be considered. Furthermore, annotations can be generated as soon as a state is calculated. They can be evaluated as soon as the immediate successors have been encountered. If the annotation evaluates to *false*, further exploration can be stopped [19]. In consequence, the process of generating S_0 can be interleaved with the process of removing states that finally leads to S. This way, memory consumption can be kept within reasonable bounds.

The number of situations in a state q of S can be reduced using, for instance, partial order reduction techniques. In ongoing research, we explore that possibility. We are further exploring opportunities for a compact representation of an operating guideline. For this purpose, we already developed a *binary decision diagram* (BDD, [20]) representation of an operating guideline for acyclic services that can be efficiently used for matching [21]. Most likely, these concepts can be adapted to arbitrary finite-state services.

We prototypically implemented our approach within the tool Fiona [10]. Among other features, Fiona can read an open workflow net and generate the operating guideline. The following example Petri nets stem from example WS-BPEL specifications of services. The WS-BPEL processes have been translated automatically into oWFN, based on the Petri net semantics for WS-BPEL [9] and the tool BPEL2oWFN [10].

The "Purchase Order" and "Loan Approval" processes are realistic services taken from the WS-BPEL specification [8]. "Olive Oil Ordering" [22], "Help Desk Service Request" (from the Oracle BPEL Process Manager) and "Travel Service" [8] are other web services that use WS-BPEL features like fault and event handling. The "Database Service" shows that it may be necessary to calculate a number of situations which is a multiple of the number of states of the considered service automaton. "Identity Card Issue" and "Registration Office" are models of administrative workflows provided by Gedilan, a German consulting company. Finally, we modeled parts of the Simple Mail Transfer Protocol (SMTP) [23]. Since it is a communication protocol, it yields the biggest operating guideline.

Table 1 provides the size of the open workflow net and the number of states of the corresponding service automaton (i. e., the inner of the oWFN), the size (number of situations, states, and edges) of the calculated operating guideline, and the time for its calculation from the given Petri net.

The examples show that operating guidelines for realistic services have reasonable size. Considering the still unexplored capabilities of reduction techniques and symbolic representations, we may conclude that the operating guideline approach is in fact feasible for tasks like service discovery (where it needs to be complemented with mechanisms for matching semantic issues).

Table 1. Experimental results running Fiona. All experiments were taken on a Intel Pentium M processor with 1.6 GHz and 1 GB RAM running Windows XP.

service *Prov*	open workflow net				inner	operating guideline			time
	places	input	output	trans.	states	situations	states	edges	(sec)
Purchase Order	38	4	6	23	90	464	168	548	0
Loan Approval	48	3	3	35	50	199	7	8	0
Olive Oil Ordering	21	3	3	15	15	5101	14	20	0
Help Desk Service	33	4	4	28	25	7765	8	10	2
Travel Service	517	6	7	534	1879	5696	320	1120	7
Database Service	871	2	5	851	5232	337040	54	147	7583
Identity Card Issue	149	2	9	114	111842	707396	280	1028	216
Registration Office	187	3	3	148	7265	9049	7	8	0
SMTP	206	8	4	215	7111	304284	362	1161	200

7 Conclusion

With the concept of an operating guideline for a service *Prov*, we proposed an artifact that can be directly executed. The operating guideline is expressed in terms of the interface of *Prov*, and gives complete information about correct communication with *Prov*. It can be manipulated in accordance with the annotations. This way, other partners can be crafted which, by construction, communicate correctly with *Prov*, too. These partners can be translated into other formalisms, most notably, oWFN.

Deciding correct interaction using an operating guideline amounts to checking the simulation relation between the partner service and the operating guideline and evaluating the annotations. It has about the same complexity as model checking deadlock freedom in the composed system itself. Due to its completeness, and due to its explicit operational structure, it can be a valuable tool in service-oriented architectures.

Experimental results have shown that the calculation of an operating guideline is feasible in practical applications.

In future work, we want to adapt the operation guideline concept to infinite-state services. However, we have strong evidence that, given an infinite-state service, the problem to construct a deadlock-freely interacting partner service is undecidable in this scenario. Besides, we want to study whether it is possible to construct operating guidelines for services without k-limited communication partners. Finally, we also want to characterize livelock-free interactions. As first examples show, it is not trivial to cover livelock-freedom by Boolean annotations.

References

1. Massuthe, P., Schmidt, K.: Operating guidelines – An automata-theoretic foundation for the service-oriented architecture. In: QSIC 2005, IEEE Computer Society (2005) 452–457

2. Massuthe, P., Reisig, W., Schmidt, K.: An Operating guideline approach to the SOA. Annals of Mathematics, Computing & Teleinformatics **1**(3) (2005) 35–43
3. Hull, R., Benedikt, M., Christophides, V., Su, J.: E-services: A look behind the curtain. In: PODS '03, ACM Press (2003) 1–14
4. Papazoglou, M.P.: Agent-oriented technology in support of e-business. Communications of the ACM **44**(4) (2001) 71–77
5. Aalst, W.v.d., Weske, M.: The P2P approach to interorganizational workflows. In: CAiSE'01. Volume 2068 of LNCS., Springer-Verlag, Berlin (2001) 140–156
6. Leymann, F., Roller, D., Schmidt, M.: Web services and business process management. IBM Systems Journal **41**(2) (2002)
7. Aalst, W.v.d.: The application of petri nets to workflow management. Journal of Circuits, Systems and Computers **8**(1) (1998) 21–66
8. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. Committee Draft, 25 January, 2007, OASIS (2007)
9. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to Petri nets. In: BPM 2005. Volume 3649 of LNCS., Springer-Verlag (2005) 220–235
10. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing Interacting BPEL Processes. In: BPM 2006. Volume 4102 of LNCS., Springer-Verlag (2006) 17–32
11. Farahbod, R., Glässer, U., Vajihollahi, M.: Specification and Validation of the Business Process Execution Language for Web Services. In: Abstract State Machines. Volume 3052 of LNCS., Springer (2004) 78–94
12. Fahland, D., Reisig, W.: ASM-based semantics for BPEL: The negative Control Flow. In: Proceedings of the 12th International Workshop on Abstract State Machines (ASM'05), Paris XII (2005) 131–151
13. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann (1996)
14. Berardi, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Mecella, M.: Automatic composition of e-services that export their behavior. In: ICSOC 2003. Volume 2910 of LNCS., Springer (2003) 43–58
15. Arias-Fisteus, J., Fernández, L.S., Kloos, C.D.: Formal Verification of BPEL4WS Business Collaborations. In: EC-Web. Volume 3182 of LNCS., Springer (2004) 76–85
16. Fu, X., Bultan, T., Su, J.: Analysis of interacting BPEL web services. In: WWW '04: Proceedings of the 13th international conference on World Wide Web, ACM Press (2004) 621–630
17. Ferrara, A.: Web services: a process algebra approach. In: ICSOC 2004, ACM (2004) 242–251
18. Badouel, E., Darondeau, P.: Theory of regions. In: Lectures on Petri Nets I: Basic Models. Volume 1491 of LNCS. (1998) 529–586
19. Weinberg, D.: Reduction Rules for Interaction Graphs. Techn. Report 198, Humboldt-Universität zu Berlin (2006)
20. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. IEEE Transactions on Computers **C-35**(8) (1986) 677–691
21. Kaschner, K., Massuthe, P., Wolf, K.: Symbolische Repräsentation von Bedienungsanleitungen für Services. In: AWPN Workshop 2006, Universität Hamburg (2006) 54–61 (in German).
22. Arias-Fisteus, J., Fernández, L.S., Kloos, C.D.: Applying model checking to BPEL4WS business collaborations. In: Proceedings of the 2005 ACM Symposium on Applied Computing (SAC), ACM (2005) 826–830
23. Postel, J.B.: Simple Mail Transfer Protocol. RFC 821, Information Sciences Institute, University of Southern California, Network Working Group (1982)