An Operating Guideline Approach to the SOA

Peter Massuthe, Wolfgang Reisig, and Karsten Schmidt

Humboldt-Universität zu Berlin
Institut für Informatik
Unter den Linden 6
D-10099 Berlin
{massuthe, reisig, kschmidt}@informatik.hu-berlin.de

Abstract Interorganizational cooperation is more and more organized by the paradigm of *services*. The *service-oriented architecture* (SOA) provides a general framework for service interaction. It describes three roles, *service provider*, *service requester*, and *service broker*, together with the three operations *publish*, *find*, and *bind*.

We provide a formal method based on Petri nets to model services and their cooperation. We characterize well-behaving pairs of requester's and provider's services and suggest *operating guidelines* as a convenient and intuitive artifact to realize *publish*. Then, the *find* operation reduces to a matching problem between the requester's service and the operating guideline. *Binding* of a requester's and a provider's service is therefore guaranteed to result in a well-behaving cooperating service.

Keywords: Services, SOA, Petri nets, Operating guidelines

1 Introduction

A *service* can be viewed as an artifact consisting of an identifier (id), an interface (e.g. specified in WSDL [4]), and internal control (e.g. a workflow). A service can typically not be executed in isolation – services are designed for being invoked by other services, or for invoking other services themselves.

The *service-oriented architecture* (SOA) [6] is a promising and increasingly influential software architecture providing a general framework for service interaction. It describes three roles of service owners: *service provider*, *service requester*, and *service broker*. A service provider *publishes* information about his service to a repository. The service broker manages the repository and allows a service requester to *find* an adequate service provider. Then, the service of the provider and the service of the requester may *bind* and start interaction.

Such cooperating services may cause non-trivial communication. Thus, for a given requester's service R, the broker's task is to select from the repository only those provided services P that are guaranteed to interact properly with R: The services R and P must not deadlock in their interaction or send unanticipated messages, for instance. Thereby, compatibility of the interfaces of R and P is not sufficient to guarantee proper interaction.

The broker must decide this task by help of the published information about P. In a currently quite popular approach, the published information is a so-called *public view* [7, 8], i.e. an abstract version P' of P with a communication behavior equivalent to P.

In this paper, we suggest an alternative: The provider does not publish information about *his* service P, but information about all proper services R of potential *requesters*, instead. This information is called *operating guideline*, OG_P, for P. In our approach, the operating guideline for P describes, in a compact way, the set of all services R that interact properly with P.

We claim that matching a requester's service R with an operating guideline OG_P is less complex than matching R with the public view P' of P. If R matches OG_P, we can guarantee that R and P interact properly. In this paper, we show that services have canonical operating guidelines and it is even possible to compute them. Furthermore, the operating guideline for P typically hides a lot of details about the internal control structure of P, that the owner of P might want to keep secret.

In our approach, we consider *workflow services*, an important subclass of services with operational behavior described as a workflow. We suggest a formal model based on Petri nets, called *open workflow nets* (oWFNs), to represent workflow services. An oWFN is basically a liberal version of a van der Aalst workflow net [1], enriched with communication places for asynchronous communication. In this paper, we present our approach only for acyclic nets.

With oWFNs, we can model services of providers, as well as services of requesters. We can furthermore compose two oWFNs and obtain a model for both services in interaction. The composition of two oWFNs results in an oWFN, again. Composition can therefore be seen as the result of the SOA *bind* operation. In our approach, we abstract from every other aspect of *bind* as resolving URI, routing, and establishment of communication channels. We assume this to be managed by an underlying middleware.

We can formulate *proper interaction* between services as a property of the corresponding composed oWFN, called *weak termination.* Each partner oWFN R that weakly terminates with P, is called a *strategy* for P.

Considering the *behaviors* of all strategies for P, it turns out that there is a unique *most permissive behavior*, i.e. *every* strategy for P has a behavior that can be obtained through restricting the most permissive one. The most permissive behavior itself then provides the basis for the operating guidelines: We can provide annotations to the most permissive behavior that characterize the allowed restrictions to obtain the behaviors of all strategies. Operating guidelines (i.e. the annotated most permissive behavior) is then provided to the service broker, thus realizing *publish.*

This way, matching a requester's service R with a published operating guideline OG_P reduces to check whether or not the behavior of R is a subtree of OG_P that satisfies the annotations.

The rest of the paper is structured as follows. In Sec. 2, we consider the essential aspects of services, and characterize the class of *workflow services.* Section 3 introduces the schema of the service-oriented architecture with the three roles for service owners and the standard operations *publish*, *find*, and *bind*.

Our model of workflow services, open workflow nets, is described in Sec. 4. This includes operational behavior, means of communication, composition, and desired properties of cooperating oWFNs.

Section 5 then introduces our main construct, operating guidelines. Operating guidelines turn out to be a convenient and elegant instrument to realize *publish*. Finally, in Sec. 6, we apply operating guidelines to decide the existence of a fitting provider's service for a given service of a requester, thus realizing *find*.

2 Services

Nowadays, cooperation across borders of enterprises is increasingly important. Functionalities are sourced out or so-called virtual enterprises for specific tasks are formed.

In this setting, *services* play an important role. A service basically encapsulates self-contained functions that interact through a well-defined interface. Recent publications apply the term service in different contexts with varying denotations. In this paper, we assume the essentials of a service to include an *identifier* (id), its *interface*, and its *operational behavior*. Thereby, the interface of a service describes means to communicate with its environment during execution. The operational behavior of a service is basically a set of operations to be executed according to some internal control structure.

The well-known class of *web services* is an implementation of services with an interface specified in WSDL and an id given by an URI.

In this paper, we concentrate on services with operational behavior described as a *workflow*, i.e. an implemented business process. Such services will be denoted as *workflow services*. Workflow services have become particularly important since the establishment of BPEL [5] as a widely excepted language to describe web services. BPEL provides control structures that typically occur in workflows.

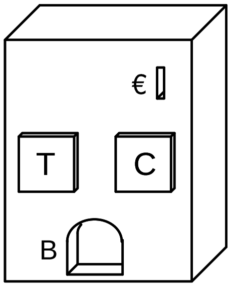

Figure 1. A vending machine that sells, for 1 Euro, either a cup of tea (button T), or coffee (button C).

Examples of workflow services include online banking systems (which are web services as well) and car rentals (which are not necessarily web services). A Java program is certainly no workflow services (but may be a web service).
As a running example, we consider the workflow service of a beverage vending machine, as outlined in Fig. 1. The service provided by this machine expects a coin (€) to be inserted and one of the buttons T and C being pressed. The service then reacts with delivering a beverage, i.e. a cup of tea (in case T has been pressed) or a cup of coffee (in case C has been pressed).

3 The Service-Oriented Architecture (SOA)

Generally, services are not executed in isolation, but in cooperation with other services (e.g. by exchanging messages). For that purpose, service interaction is organized by the SOA. The SOA assumes that services are run by agents, with agents entering (and leaving) the scene dynamically. The services of these agents are intended to communicate with each other. This requires an agent to establish new communication facilities with other agents (in particular in case the agent entered the scene only recently).
In the SOA, such communication facilities are established by help of a *service broker*. Each agent is assumed to approach the broker in one of two roles: As a *service provider*, i.e. in the role of delivering some service, or as a *service requester*, i.e. in the role of using an already provided service.

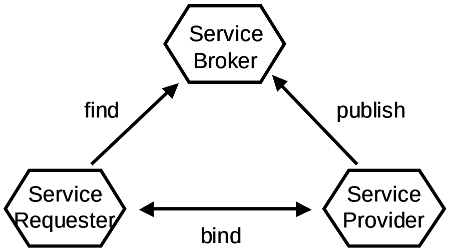

Figure 2. The service-oriented architecture (SOA).

Therefore, the provider, requester, and broker agents execute the following three operations:
A service provider sends information to the service broker, indicating how a service requester may use his service. The service broker then stores this information together with the provider's id in a repository. This operation is called *publish*.
The SOA operation *find* means that a requester sends a description of his requested service to the service broker. The broker selects a fitting one and returns the corresponding provider's id.

Finally, the requester establishes a connection with the provider, and both agents jointly run their respective services, described by the SOA *bind* operation.

The three roles of agents, together with the three standard operations are outlined in the SOA triangle, depicted in Fig. 2.

The above operations come up with a number of problems:

Publish: The provider has good reasons to keep published information about his service at a minimum. He usually wants (1) to cover business secrets, (2) to retain maximal flexibility to update his service without giving notice to providers and brokers, and (3) to shield requesters from details they do not need to know.

In this paper, we suggest operating guidelines (to be introduced in Sec. 5) as information to be published about a provider's service. Operating guidelines serve well as an abstraction from internal details and support flexibility.

Find: Given a service of a requester and the operating guideline for a provider, a broker has to decide whether or not the requester's and the provider's services would interact properly.

In this paper, we describe how a broker may decide this question by matching the requester's service with operating guidelines.

Bind: Through our operating guidelines approach to *publish* and *find*, a requester's service is guaranteed to successfully cooperate with the service of a broker's recommended provider (e.g. they do not deadlock). We completely abstract from implementation details concerning the establishing of communication channels between provider and requester (such as resolving an URI, routing, etc.). We just suggest means to describe the behavior of single services as well as their cooperation.

Summing up, the SOA schema requires a proper representation of services and their cooperation, together with adequate descriptions of the operations *publish* and *find*.

The rest of this paper suggests corresponding features for the subclass of services, called workflow services.

4 Models of Workflow Services

A solution to the problems described above requires a proper model of workflow services. A model of workflows was already suggested by van der Aalst [1]. He defines a special class of Petri nets, *workflow nets* (WFNs), that adequately describe the control structure of workflows. Since workflow services are supposed to communicate with other workflow services, additional constructs for modeling communication channels are needed. We suggest *open workflow nets* (oWFNs) for this endeavour, essentially a liberal version of van der Aalst workflow nets, enriched with communication places. Each communication place of an oWFN models a channel to send (receive) messages to (from) another oWFN. Thereby, we abstract from data and just model the occurrence of messages as undistinguishable tokens.

We assume the usual representation of Petri nets $N = (P, T, F)$, with P and T the set of *places* and *transitions* (graphically, circles and squares), and a set $F \subseteq (P \times T) \cup (T \times P)$ of *arcs*, graphically: arrows. A *marking* is a mapping $m : P \to \mathbb{N}$ (graphically, $m(p)$ black tokens on p). As usual, a transition t is *enabled* at a marking m if for each place p with $(p,t) \in F$, $m(p) \geq 1$.

If enabled at m, *occurrence* of t then yields the marking m' with $m'(p) = m(p) - 1$ if $(p,t) \in F$ and $(t,p) \notin F$, $m'(p) = m(p) + 1$ if $(t,p) \in F$ and $(p,t) \notin F$, and $m'(p) = m(p)$ otherwise.

An *open workflow net* is a Petri net $N = (P,T,F)$ together with

1. two sets $in, out \subseteq P$, such that for all transitions $t \in T$ holds: if $p \in in$ $(p \in out)$ then $(t,p) \notin F$ $((p,t) \notin F)$,
2. a distinguished marking m_0, called the *initial marking*, and
3. a set Ω of distinguished markings, called the *final markings* of N.

The places in *in* (*out*) are called *input* (*output*) places. The set $in \cup out$ is called the *interface* of N. The *inner of* N can be obtained from N by removing all interface places, together with their adjacent arcs.

Graphically, N is surrounded by a dashed box, with the interface places on its boundary. As a convention, we label a transition t connected to an input (output) place x with $?x$ $(!x)$.

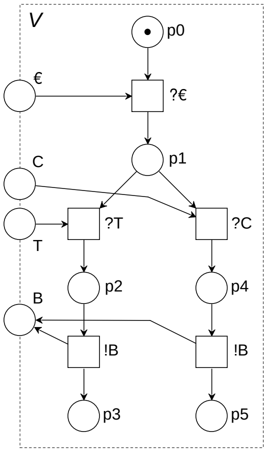

Figure 3. An oWFN V for the vending machine of Fig. 1.

As an example, Fig. 3 shows an oWFN, V, modeling the vending machine of Fig. 1. The places €, T, C, and B denote an inserted coin, the button T or C pressed, and a beverage delivered, respectively. There are two final markings of V: a single token on p3 or a single token on p5.

We are now ready to define the composition of oWFNs, reflecting the interplay between workflow services.

Conceiving the vending machine of Fig. 1 as a provider's service, a corresponding customer would insert a coin, press one of the buttons and later on receive the beverage.

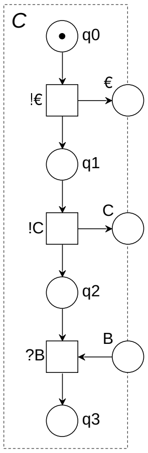

Figure 4. A customer's oWFN, C, for the vending machine V that wants coffee.

Fig. 4 models a customer, C, of the vending machine, pressing the *coffee* button. This model is again an oWFN. Further examples can be found in [12].

The interaction of two oWFNs is reflected by their *composition*. Two oWFNs M and N may have some elements in common. Nevertheless, without loss of generality we may assume that M and N only share input- and output elements:

$$(P_M \cup T_M) \cap (P_N \cup T_N) \subseteq (in_M \cup out_M) \cap (in_N \cup out_N) \qquad (1)$$

This property in fact holds for the two oWFNs V and C: They share nothing but the interface places €, C, and B.

Assuming (1), composition of two oWFNs M and N is an oWFN again, denoted $M \oplus N$, and constructed essentially as the component-wise union of M and N. So let $M \oplus N$ be defined by $P_{M\oplus N} =_{def} P_M \cup P_N$, $T_{M\oplus N} =_{def} T_M \cup T_N$, $F_{M\oplus N} =_{def} F_M \cup F_N$.

Each place in $out_M \cap in_N$ (or in $in_M \cap out_N$) turns into an inner place of $M \oplus N$. With $I =_{def} (out_M \cap in_N) \cup (in_N \cap out_M)$, let $in_{M\oplus N} =_{def} (in_M \cup in_N) \setminus I$ and $out_{M\oplus N} =_{def} (out_M \cup out_N) \setminus I$.

For markings m_M and m_N of M and N, respectively, let $m_M \oplus m_N$ be a marking of $M \oplus N$, defined for $p \in P_{M\oplus N}$ by $(m_M \oplus m_N)(p) =_{def} m_M(p) + m_N(p)$, where $m_M(p) = 0$ if $p \notin P_M$ and $m_N(p) = 0$ if $p \notin P_N$. Then, let $m_{(M\oplus N)_0} =_{def} m_{M_0} \oplus m_{N_0}$ and $m_{M\oplus N} \in \Omega_{M\oplus N}$ iff $m_{M\oplus N} = m_M \oplus m_N$ for some $m_M \in \Omega_M$ and some $m_N \in \Omega_N$.

As an example, Fig. 5 shows the oWFN $C \oplus V$. This oWFN has two terminal markings, m_1 and m_2, with $m_1(\mathsf{q3}) = m_1(\mathsf{p3}) = 1$, $m_2(\mathsf{q3}) = m_2(\mathsf{p5}) = 1$, and no tokens on all other places. Notice that $in_{C\oplus V} = \{\mathsf{T}\}$ and $out_{C\oplus V} = \emptyset$.

Fig. 6 shows another oWFN, E. Assume one terminal marking for E, with a token on r2 and no token elsewhere. E models an erroneous customer service of the vending machine, as the customer apparently „forgets“ to

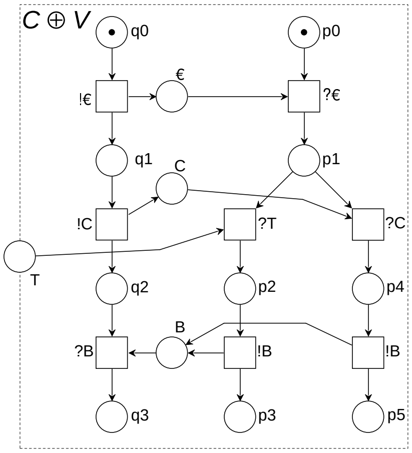

Figure 5. The composed oWFN $C \oplus V$ of Fig. 3 and Fig. 4.

press one of the machine buttons, and both services deadlock. Summing up, the oWFN C is an „adequate“ partner for V, whereas E is not.

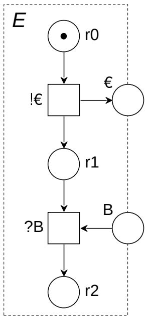

Figure 6. An erroneous partner oWFN E for V.

In technical terms, a marking m of an oWFN is a *deadlock* if m enables no transition at all. It is easy to see that in the composed oWFN $C \oplus V$, the only reachable deadlock is a final marking. In contrast, in the oWFN $E \oplus V$ (not shown here), the marking m with $m(\mathsf{r1}) = m(\mathsf{p1}) = 1$ and $m(p) = 0$ for all other places $p \in P_{E \oplus V}$ is a reachable deadlock which is no final marking.

An oWFN in which all deadlocks are final markings is called *weakly terminating*. Given an oWFN N, we call an oWFN M a *strategy* for N iff the oWFN $N \oplus M$ is weakly terminating. For example, C is a strategy for V, whereas E is not.

5 Publish

As mentioned earlier in this paper, information published by a service provider on his service P must enable the service broker to decide whether or not a requester's service R is a strategy for P (otherwise, binding P with R may result in unexpected behavior). Whether or not R is a strategy for P depends on the internal control structure of P. Hence, it has been proposed to publish a public view P' of P to the service broker. It is supposed that, by knowing P', the broker can decide whether or not R will interact properly with P.

We propose a different approach: Instead of a description of the provided service P, we suggest to publish a description of the set of all *strategies* (i.e. all properly interacting oWFNs) R for P, directly. In fact, we provide a description of the *behaviors* of all strategies R. We call this description operating guideline for P and write OG_P.

In the remainder of this section, we give brief answers to the following questions for a given oWFN P: (1) What is a behavior? (2) How does the operating guideline OG_P look like? (3) How can it be computed? (4) Why does it cover all strategies for P?

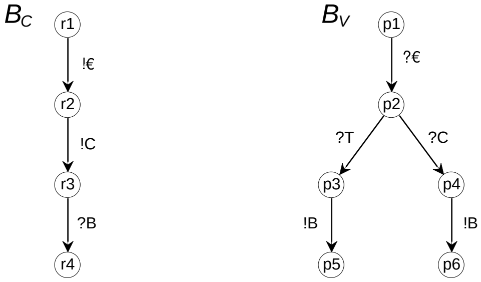

Figure 7. The behaviors B_C and B_V of the oWFNs C and V, resprespectively.

The *behavior* of a usual Petri net can be represented as a reachability tree. This notion is, however, not adequate for oWFNs, since the marking on the interface places can be changed by the environment. Thus, we describe the behavior of an oWFN N by a slightly different structure. We first compute the reachability tree of the *inner* of N (see Sec. 4). Due to our restriction to acyclic oWFNs, the reachability tree is finite. Then, each edge in the reachability tree is annotated with $!x$ ($?x$) if the corresponding transition in N is connected to an output (input) place x, and with τ, otherwise. This answers question (1) stated above.

Figure 7 shows the behaviors B_C and B_V of the oWFNs C and V of Fig. 4 and Fig. 3, respectively.

In the following, to answer question (2), we first present the two ingredients to operating guidelines for P: the most permissive behavior of strategies for P and annotations how to derive the behaviors of all strategies. Then, we sketch the algorithms to compute both ingredients, answering questions (3) and (4).

These answers rely on results proven in [13, 11] concerning behaviors of strategies. For the purpose of simplicity only, we constrain the considerations in this paper to *deterministic* behaviors of strategies. A behavior is deterministic iff every edge of the behavior has exactly one expression $!x$ or $?y$ attached (i.e. there is no silent move τ), and there is no node in the behavior where two leaving edges have the same expression attached. All behaviors shown in this paper are deterministic.

Let the set of the (deterministic) behaviors B_R of *all* strategies R for P be denoted by $\mathcal{B}_\mathcal{P}$. We can establish a (partial order) relation, *more permissive*, to behaviors of $\mathcal{B}_\mathcal{P}$: A behavior B is more permissive than a behavior B' if B' is isomorphic to a subtree of B containing the root.

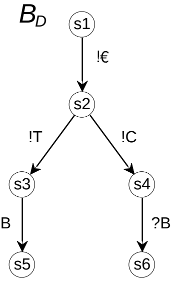

Figure 8. A more permissive behavior B_D of a customer, who decides.

As an example, Fig. 8 shows the behavior B_D of some customer D of the vending machine, who first inserts a coin and then *decides* for coffee or tea. B_D is more permissive than B_C, whereas neither B_C is more permissive than B_V, nor B_V is more permissive than B_C.

In [13, 11], we show that, for every oWFN P, the set $\mathcal{B}_\mathcal{P}$ has a unique most permissive element, the *most permissive behavior*, B^*. Consequently, we call every oWFN R with the most permissive behavior as its behavior (i.e. $B_R = B^*$), a *most permissive strategy* for P. While the most permissive *behavior* is unique, most permissive strategies are not. There are typically many structurally different Petri nets that share the same behavior. In particular, our presented concept of behavior does not distinguish arbitrary interleaved transitions from truly concurrent transitions. The main property of the most permissive behavior B^* is that it comprises all behaviors of strategies for P: Every behavior B_R of a strategy R for P is (isomorphic to) a subtree of B^*. Thus, the most permissive behavior serves as the first ingredient to the operating guideline for P.

Unfortunately, the converse is not true. Not every subtree of the most permissive behavior is itself a behavior of a strategy. Thus, the remaining problem is to distinguish those subtrees of the most permissive behavior which are behaviors of strategies from those subtrees which are no behaviors of strategies. Our solution to this task is again based on a result proven in [13, 11]:

Given a provided oWFN P and a behavior B_R of some requester's service R, we can decide for each node q_R of B_R whether or not it can cause a deadlock in $R \oplus P$. This is basically determined by the edges that leave q_R: Whether or not R is a strategy for P depends on present or missing edges in B_R. Thus, we code the constraints for edges leaving q_R as a boolean formula over edge labels and annotate it to q_R. B_R satisfies these constraints if and only if R is a strategy.

Since the most permissive behavior B^* is a behavior of some strategy, we can annotate B^*, too. A subtree of B^* is thus a behavior of a strategy if and only if it still satisfies the attached annotations. The annotations to the nodes of B^* constitute the second ingredient and complete the operating guideline for P. This answers question (2) stated above.

As an example, the operating guideline OG_V for the vending machine V (of Fig. 3) is depicted in Fig. 9. The possibility to first press a button and then inserting a coin comes from the proposed asynchronous communication. The annotations to nodes which have only one outgoing edge are skipped.

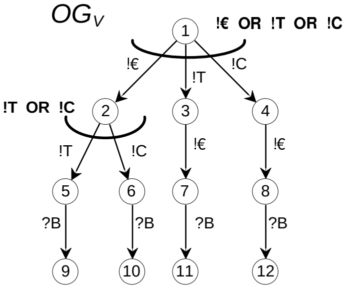

Figure 9. The operating guideline OG_V for the vending machine V.

The rest of this Section is devoted to questions (3) and (4). We sketch the algorithms to construct the most permissive behavior and to annotate the nodes and argue why operating guidelines cover all behaviors of strategies.

The most permissive behavior for an oWFN P can be constructed as follows. Consider, as a starting point, the complete, deterministic behavior B^C of some partner oWFN R where for every node and every input (output) place x of P, there is an outgoing edge labeled $!x$ ($?x$). We can limit the depth of B^C by the depth of the behavior B_P of P. Obviously, every behavior B_R of a strategy for P is a subtree of B^C.

Next, we compute a transition system from the composition of B^C with B_P. A state of the composed system consists of a state q of B^C, a state p of B_P, and a multiset (bag) M of pending messages.

Then, we remove, in an iterative process, those nodes q from B^C for which there is a bad node $[q, p, M]$ in the composed system, i.e. a node which represents an undesired situation like a deadlock that is no final marking, or unconsumable pending messages. For the removed states, we can show that they cannot be part of the behavior of any strategy for P. Thus, every behavior of strategies remains a subtree of B^C obtained after each iteration. Finally, we can show that as soon as the iterative process of removing states terminates, the remaining behavior (if any) is in fact the behavior of a strategy - and even the most permissive behavior B^*. This problem is, in fact, a problem known in the literature as *controller synthesis* [3, 10].

To annotate the nodes of B^*, consider again the transition system composed from B^* and B_P. In that tree there is no bad node (since it would have been removed by the above algorithm). Removing further nodes (with adjacent edges) in B^* results in removing multiple nodes (and edges) in the composed system. This may cause deadlocks or pending messages are not consumed anymore. Hence, other nodes in the composed system may become bad nodes. If this happens, the considered node must not be removed. This can easily be expressed by a boolean formula over the labels of outgoing edges. These formulae are attached to each node in B^* and express exactly, which subtrees of B^* are strategies itself. This completes the answers to the questions (3) and (4).

To summarize, the operating guideline for an oWFN P can be constructed automatically and only by knowing P. As we assume the construction is done by the owner of P, this is acceptable. Furthermore, the annotations just reflect needed actions of the environment such that the composed system does not deadlock. The annotations do not reflect *why* a deadlock can be reached, nor how the deadlock looks like. When published, operating guidelines therefore hide most details about the internal control structure of the provided service, that the service provider might want to keep secret.

The presented algorithm to construct the most permissive behavior is just a theoretic algorithm that comes from a constructive proof of the existence of B^* in the set $\mathcal{B}_{\mathcal{P}}$. Its starting point, the complete, deterministic behavior B^C may become rather large for realistic services. Even larger is the composed system of B^C and B_P, needed to characterize the nodes of B^C.

In current research, we develop efficient representations of operating guidelines as binary decision diagrams (BDDs) [2], as well as the construction of operating guidelines directly as a BDD. BDDs are a data structure that proved to be capable of representing large transition systems in the area of model checking. Preliminary results in the application of BDDs representing operating guidelines are promising.

6 Find

Matching a requester's service with an operating guideline OG_P is rather simple. Given an oWFN R of the requester, we first compute its behavior, B_R. This is simple and well-understood state space generation. Then, we

need to check whether B_R (a) is isomorphic to a subtree of OG_P and (b) satisfies the annotations. Thereby, a literal $?x$ ($!x$) at some node of OG_P is evaluated to *true* if there is an outgoing edge from the corresponding node in B_R labeled $?x$ ($!x$) and evaluated to *false*, otherwise.
It is easy to see that B_C and B_D match OG_V, whereas the behavior B_E of the erroneous partner oWFN E would not match OG_V.
Checking the subtree property can be solved by a coordinated depth-first search through both behaviors. Its run-time is linear in the size of R's behavior. Checking the annotations amounts to computing a value of a boolean formula and can thus be implemented efficiently. Thus, the *find* procedure based on operating guidelines turns out to be very efficient.
In contrast, a *find* based on public views is a more complex operation. Given a requesting service R and a public view P' of a provided service P, a service broker must decide whether R is a strategy for P. Currently, the only available approach to this problem is to build the system composed of P' and R and to check its state space for deadlocks and end states with unconsumed messages. The size of the state space is typically in the same order of magnitude as the number of states of P' times the number of states of R. Checking the state space for deadlocks is linear in that number and thus more complex than matching R with OG_P.
The reader might argue that a more complex *find* may be compensated by the fact that public view generation is less complex than generating operating guidelines. We cannot verify statements about the complexity of public view generation, since we do not know any convincing solution to public view generation. The only formal approach known to us is [8] where a few sound abstraction rules for services are proposed which preserve strategies. It is, however, unclear whether the generated public views satisfy the requirement of being sufficiently distinct from the original service to keep the service itself secret. Besides these doubts, an efficient *find* is still more important than an efficient *publish*. The *publish* operation is performed once by a provided service. *Find* instead, that is, matching a requesting service against a public view or an operating guideline, is performed many times. This is due to the fact that one and the same requesting service must be checked against many provided services before a matching service can be found. We thus believe that the operating guidelines approach is more suitable to the SOA than the public view approach. For a more detailed discussion on public views and operating guidelines see [9].

7 Conclusion

We propose oWFN as a formal model for services that use workflows as their internal control structure. We showed that it is possible to automatically compute, for an oWFN P, an operating guideline OG_P which characterizes the set of all (deterministic) behaviors of strategies for P. We propose to use OG_P as information published in service repositories. This way, it is easy for the service broker to assign well-behaving pairs of provider's and requester's services: the requester's service must match the operating guideline published for the provided service. Generating an

operating guideline may be complex, but we expect that this complexity can be managed through the use of advanced technology developed in the area of model checking. In turn, matching a service with an operating guideline is considerably simpler than checking compliance between a requester's service and a public view of a provided service.
We see several directions for future research. First, we need to extend the approach to services containing cycles. We have a number of preliminary results on this matter. Second, we study specialized operating guidelines, characterizing, e.g., the set of all those strategies of the considered vending machine that inevitable lead to the delivery of coffee. Third, we investigate further important aspects which are relevant for selecting a service such as real-time constraints or cost models. We want to extend the concept of operating guidelines to those aspects.
We are convinced that our approach is well suited to implement the service discovery outlined in the SOA triangle. Our concept is quite close to those guidelines that are attached to real vending machines. The concept of operating guidelines has thus been already successful in every-day life for a long time.

References

1. W. M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
3. C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer Academic Publishers, 1999.
4. Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weeravarana. Web Service Discription Language (WSDL) 1.1. Technical report, Ariba, International Business Machines Corporation, Microsoft, March 2001.
5. F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Specification, BEA Systems, IBM, Microsoft, SAP, Siebel, 05 May 2003. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbiz2k2/html/bpel1-1.asp.
6. Karl Gottschalk. Web Services Architecture Overview. Ibm whitepaper, IBM developerWorks, 01 September 2000. http://ibm.com/developerWorks/web/library/w-ovr/.
7. F. Leymann, D. Roller, and M. Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2), 2002.
8. A. Martens. *Verteilte Geschäftsprozesse - Modellierung und Verifikation mit Hilfe von Web Services.* PhD thesis, Institut fr Informatik, Humboldt-Universität zu Berlin, 2004.
9. P. Massuthe and K. Schmidt. Operating guidelines - an alternative to public view. Techn. Report 189, Humboldt-Universität zu Berlin, 2005.

10. P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization*, 25(1):206–230, 1987.
11. W. Reisig, K. Schmidt, and Ch. Stahl. Kommunizierende Geschäftsprozesse modellieren und analysieren. Accepted for *Informatik – Forschung und Entwicklung*, 2005.
12. Wolfgang Reisig. Modeling- and analysis techniques for web services and business processes. In *FMOODS*, volume 3535 of *Lecture Notes in Computer Science*, pages 243–258, 2005.
13. K. Schmidt. Controllability of business processes. Techn. Report 180, Humboldt-Universität zu Berlin, 2004.