

Operating Guidelines for Services

Peter Massuthe¹ and Karsten Wolf²

¹ Humboldt-Universität zu Berlin
Institut für Informatik
Unter den Linden 6
D-10099 Berlin
massuthe@informatik.hu-berlin.de

² Universität Rostock
Institut für Informatik
D-18051 Rostock
karsten.wolf@informatik.uni-rostock.de

Abstract. In the service-oriented architecture (SOA), we distinguish three roles of service owners: service providers, service requesters, and service brokers, and the three standard operations *publish*, *find*, and *bind*.

We provide a formal method based on Petri nets to model services. We 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.

1 Introduction

A *service* is an artifact that consists of an interface and internal control. The *service-oriented architecture* (SOA) [2] provides a framework for the interaction of services. It distinguishes three roles of service owners: *service provider*, *service broker*, and *service requester* (see Fig. 1). It postulates a general protocol for interaction: A provider registers at the service broker by submitting information about how to interact with its service. The service broker manages such information about all registered service providers 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.

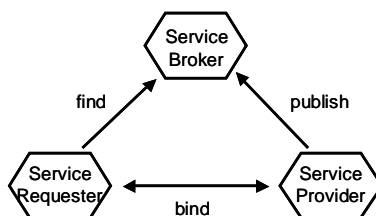


Fig. 1. The service-oriented architecture.

The interaction of services may cause nontrivial communication between a requester and a provider. Consider, as a running example for a provided service, a vending machine as depicted in Fig. 2. If this service is bound to a requester's service, the requester must send a coin (€), press a button (T or C), and finally receive a beverage (B).

It is obviously desirable that a service requester gets assigned only such a service provider that their services do not ill-communicate with each other (such as running into a deadlock or sending unanticipated messages). In our example, the broker must not deliver our vending machine service to a requester that wants to pay in other currencies than € or a requester who expects the beverage before paying.

For this purpose, the service broker needs information about the internal control structure of the provider's service – the provider's interface only (e.g. a WSDL specification) is not sufficient. Publishing the whole internal control to the service broker would solve the problem. This is, however, not feasible, as the service provider may want to keep its internal structure secret.

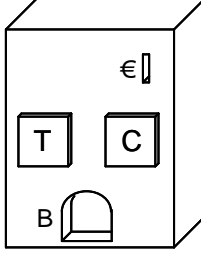


Fig. 2. A vending machine that sells, for 1 Euro, either a cup of tea (press T), or a cup of coffee (press C).

We propose the published information to be an *operating guideline* for the provider’s service P . The operating guideline for P essentially represents, in a condensed form, the set of *all* well-communicating services R of requesters of the service P . Thus, the operating guideline for the vending machine in our example would cover requesters that pay in €, but not a requester who pays in £, for instance.

In our operating guidelines approach, the broker’s task to select a fitting provider for a querying requester reduces to a matching problem. This is basically a check whether the requester’s service “follows” the published operating guideline.

2 Models of Workflow Services

We suggest *open workflow nets* (oWFNs) to model services. Open workflow nets are a special class of Petri nets and can be seen as 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. *Workflow modules* as introduced in [3] are a very similar Petri net class.

Formally, an *open workflow net* is a place transition Petri net $N = (P, T, F)$ together with

- two sets $in, out \subseteq P$, such that for all transitions $t \in T$ holds: if $p \in in$ ($p \in out$) then $(t, p) \notin F$ ($(p, t) \notin F$),
- a distinguished marking m_0 , called the *initial marking*, and
- 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 . As a convention, we label a transition t connected to an input (output) place x with $?x$ ($!x$). The *inner* of N can be obtained from N by removing all interface places, together with their adjacent arcs. In this paper, we present our approach only for acyclic nets, i. e. nets where the transitive closure of F contains no cycles.

As an example, Fig. 3(b) shows the oWFN V , modeling the vending machine of Fig. 2. 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.

A corresponding customer would insert a coin, press one of the buttons and later on receive the beverage. The oWFN C , depicted in Fig. 3(a), models a customer of the vending machine, pressing the *coffee* button.

The interaction of two oWFNs is reflected by their *composition*. The 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 .

As an example, Fig. 4 shows the oWFN $C \oplus V$. This oWFN has two terminal markings, m_1 and m_2 , with $m_1(q3) = m_1(p3) = 1$, $m_2(q3) = m_2(p5) = 1$, and no tokens on all other places. Notice that $in_{C \oplus V} = \{T\}$ and $out_{C \oplus V} = \emptyset$.

Now consider another customer’s oWFN E (depicted in Fig. 5) where $!€$ and $?B$ are executed in sequence. E models an erroneous customer service of the vending machine, as the customer apparently “forgets” to press one of the machine buttons, and both services deadlock. The oWFN C represents an “adequate” partner for V , whereas E is not “adequate” 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 oWFN $C \oplus V$, the only reachable deadlock is a final marking. In contrast, in $E \oplus V$ (not shown here), the marking m with $m(r1) = m(p1) = 1$ and $m(p) = 0$ for all other places $p \in P_{E \oplus V}$ is a reachable deadlock which is no final marking.

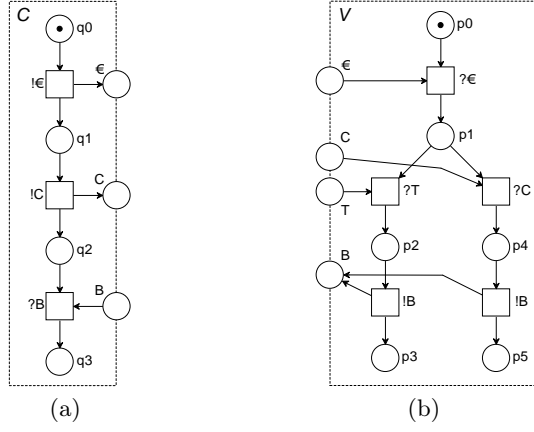


Fig. 3. An oWFN V modeling the vending machine (right), together with an oWFN C modeling a customer who wants coffee (left).

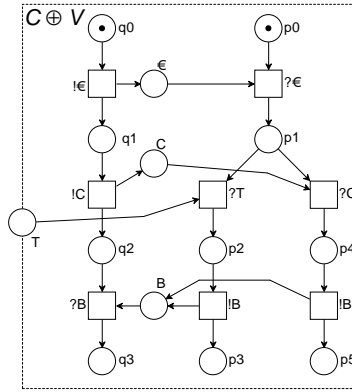


Fig. 4. The composed oWFN $C \oplus V$ of Fig. 3(b) and Fig. 3(a).

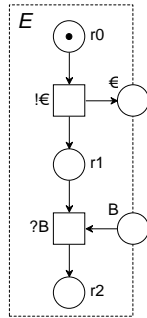


Fig. 5. An erroneous partner oWFN E for V .

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

3 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 . Thus, we suggest to publish directly a description of the set of all *strategies* (i.e. all properly interacting oWFNs) R for P . 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 .

The behavior of an oWFN N is the reachability tree of the inner of N , where transitions are annotated with $!x$ ($?x$) if they put a token on (take a token from) an interface place. For example, the behaviors B_C and B_V of the oWFNs C and V of Fig. 3(a) and Fig. 3(b) are depicted in Fig. 6(a) and Fig. 6(b), respectively.

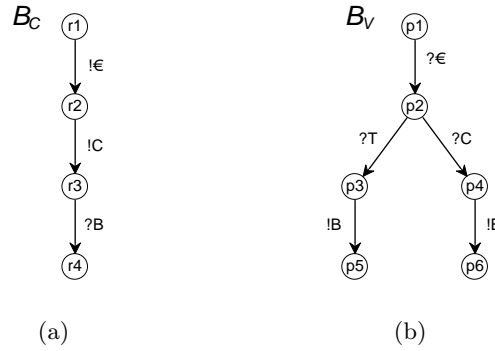


Fig. 6. The behaviors B_C and B_V of the oWFNs C and V , respectively.

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

As an example, Fig. 7 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. Obviously, B_D is more permissive than the behavior B_C of the coffee customer.

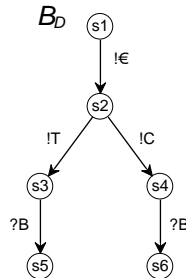


Fig. 7. A more permissive behavior B_D of a customer, who decides.

In [7,6], we show that, for every oWFN P , the set \mathcal{B}_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 .

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 [7,6]:

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 .

As an example, the operating guideline OG_V for the vending machine V (of Fig. 3(b)) is depicted in Fig. 8. The possibility to first press a button and then inserting a coin comes from the proposed asynchronous communication.

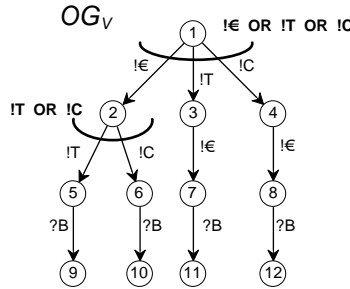


Fig. 8. The operating guideline OG_V for the vending machine V . Annotations of nodes with only one outgoing edge are skipped.

The operating guideline for an oWFN P can be constructed automatically. Construction algorithms can be found in [7]. The construction bases on the provider's oWFN P , only. As we assume the construction is done by the provider himself, 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.

In current research, we develop efficient representations of operating guidelines as binary decision diagrams (BDDs) [1]. 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.

4 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.

As an alternative to operating guidelines, the concept of *public views* [4,5] has been proposed. It suggests to publish an abstract version P' of a process P to the service broker. A *find* based on public views is more complex than a *find* based on operating guidelines: 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 .

5 Conclusion

We propose oWFNs 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. Randal E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.
2. Karl Gottschalk. Web Services architecture overview. IBM whitepaper, IBM developerWorks, September 2000. <http://ibm.com/developerWorks/web/library/w-ovr/>.
3. Ekkard Kindler, Axel Martens, and Wolfgang Reisig. Inter-operability of Workshop Applications - Local Criteria for Global Soundness. In Wil M. P. van der Aalst, Jörg Desel, and Andreas Oberweis, editors, *Business Process Management*, volume 1806 of *Lecture Notes in Computer Science*, pages 235–253. Springer-Verlag, 2000.
4. F. Leymann, D. Roller, and M. Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2), 2002.
5. A. Martens. *Verteilte Geschäftsprozesse - Modellierung und Verifikation mit Hilfe von Web Services*. PhD thesis, Institut für Informatik, Humboldt-Universität zu Berlin, 2004.
6. W. Reisig, K. Schmidt, and Ch. Stahl. Kommunizierende Geschäftsprozesse modellieren und analysieren. Accepted for *Informatik – Forschung und Entwicklung*, 2005.
7. K. Schmidt. Controllability of business processes. Techn. Report 180, Humboldt-Universität zu Berlin, 2004.