Filtering Undesirable Service Substitution Behaviors using Filtering Guidelines

Jarungjit Parnjai

Humboldt-Universität zu Berlin, Institut für Informatik, Berlin, Germany.
parnjai@informatik.hu-berlin.de

Abstract. A service T can substitute a service S if T interacts with every partner of S in a deadlock-free manner. We introduce the notion of a *filtered service* of a service S with less or the same traces of external messaging behavior as S, yet can substitute S. We propose a finite representation of all *filtered* services of S called *filtering guidelines*.
Given a service T that cannot substitute S, we can employ the filtering guidelines based techniques to construct a filtered service for S from T by *filtering* out certain undesirable behavior of T that are not described by the filtering guidelines.

1 Introduction

Service-orientation supports process evolution by considering a complex business process as a collaboration of several simpler, interacting services. Substituting one or more of these services by another may endanger the proper interaction in unexpected ways.

In this paper, we study the evolution of the business protocols [7] which specifies the external messaging behavior that are exchanged between stateful services. We consider the behavioral substitution criterion that is formalized by the notion of *accordance* [9]. A service T can substitute a service S under accordance if T can interact with every partner R (in every context) of S in a deadlock-free manner.

Whenever a service S changes due to various reasons, e.g. changes in regulations or the operational behavior of services, an incremental modification of an existing protocol requires a construction of new service T without the need of redefining T from scratch. Yet, constructing a new service T that can substitute a service S is a time-consuming and error-prone task, typically based on trial-and-error methods. The activity to construct a substitutable service requires a systematic support from formal approaches and tools that naturally optimize the time and effort to construct such a service.

Nevertheless, the languages and tools that are currently available on the market offer only limited support. The language WS-BPEL, for example, has rules (called profiles) allowing to transform a service S into a service T that can substitute S. These syntactical rules are usually restricted and their extensions regarding behavioral compatibility (e. g., in [1]) are still incomplete, and hence it is not possible to construct every service that can substitute a given service S by means of transformation.

To systematically support such a construction, a finite representation of all services that can substitute a given service has been proposed in [6, 8]. Such a finite representation describes all services that can substitute a given service S under accordance, and therefore, this approach realizes several analysis and synthesis challenges of service substitution such as deciding and constructing a service that can substitute S under accordance.

Consider a service T that cannot substitute S under accordance, this means T is not describing by a finite representation proposed by [6, 8]. In many situations, a new service that is constructed from such a representation introduces either additional external message behaviors (e. g., a new order of sending message activities) or completely new behavior where none of behavior of T is preserved. Nevertheless, we may want to promote reusability and rapid development of services by *removing* undesirable behavior of T rather than introducing new behavior, and by doing so, we derive a new service T' from T such that T' can substitute S under accordance. Such a *filter* operation on service behaviors should allow the construction of a new service T', that can substitute S under accordance, from service T with optimal amount of time and effort.

In this paper, we extend the approach of [6, 8] by presenting *filtering guidelines* that realizes the *filter* operation for services. The *filtering guidelines* for service S describes all services that have less or the same traces of external messaging behavior as S, yet can substitute S under accordance. With our approach, one can construct a new *filtered service* using the filter guidelines to *filter out* the undesirable behavior that possibly introduces a deadlock when interacting with a deadlock-free partner of S. We ensure that the *filtered service* can substitute service S under accordance.

The remainder of this paper is organized as follows. Section 2 describes preliminary notions and related works. Section 3 defines filtering services and presents a finite representation of all filtering services. Finally, Section 4 concludes the paper and sketches possible extensions of the approach and future work.

2 Background and Related Works

A service consists of a control structure describing its *behavior* and an interface for asynchronous communication with other services. An interface is a set of (input and output) *channels*. We abstract from the syntax of service description languages and use *service automata* to model service behavior.

A *service automaton* [4] (or *service*, for short) consists of a finite set Q of states, an initial state $q_0 \in Q$, a set I of input channels, a set O of output channels (I and O are disjoint and do not contain internal message τ), a non-deterministic transition relation $\delta \subseteq Q \times (I \cup O \cup \{\tau\}) \times Q$, and a set Ω of finals states. A non-final state with no outgoing transition is a *deadlock*.

Example 1. Figure 1 shows the communication skeleton (or abstract processes in terms of BPEL) of three customer services S, T_1, and T_2 as service automata.

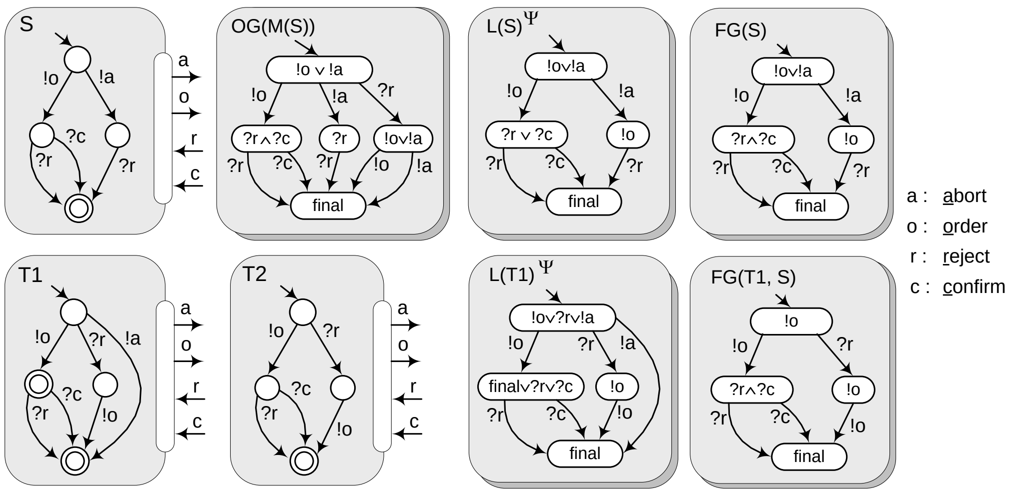

Fig. 1. Running examples

All customers have the same interface ($I = \{r, c\}$ and $O = \{a, o\}$) and can send an order (labeled $!o$), send an abort message (labeled $!a$), receive a reject message (labeled $?r$), or receive a confirm message (labeled $?c$). Each customer has different behavior described by its control flow. Final states are depicted with double circles.

We assume asynchronous communication between services with bounded and unordered message buffer (i. e., messages may overtake each other in a bounded channel). Two services R and S can be *composed* if they have compatible interface (input channels of R are the output channels of S, and vice versa). The composition yields a service with empty interface. Two services R and S interact properly if their composition is *deadlock-free* (every reachable non-final state has an outgoing transition). In that case, R is a *controller* (i. e., deadlock-free partner) of S. The set *Controllers*(S) denotes the set of all controllers of S. The notion of a controller is symmetric; if R is a controller of S, then S is also a controller of R. If S has at least one controller, then S is *controllable.*

A service T can substitute a service S under *accordance* [9] if (1) S and T have the same interface (same input and output channels) and (2) every controller of S is a controller of T (i. e., $\mathit{Controllers}(S) \subseteq \mathit{Controllers}(T)$). The set $\mathit{Accord}(S)$ denotes the set of all services T that *accords with* S (i. e., T can substitute a service S under accordance).

Example 2. In Fig. 1, service T_1 cannot substitute service S under accordance. This is because T_1 terminates after sending abort message, but there is one deadlock-free partner of S (not shown in the examples) which, after receiving abort message, will send back a reject message. Obviously, T_1 cannot interact deadlock-freely with this partner of S. However, T_2 can substitute S under accordance, because T_2 can interact deadlock-freely with every partner of S.

Each controllable service S has a *most-permissive controller* of S, denoted by $mp(S)$, which can exhibit all behaviors that any controller of S can exhibit (i. e., if $R \in Controllers(S)$, then $mp(S)$ must simulate R [10]). The set of all controllers of a controllable service S can be represented by the *operating guidelines* $OG(S)$ of S [4], denoted by $OG(S)$. The operating guidelines $OG(S)$ of S, is the Boolean annotated most-permissive controller $mp(S)^{\phi}$ of S, where each state q of $mp(S)$ of S is annotated with a Boolean formula $\phi(q)$. These formulas consist of conjunctions $\wedge$, disjunctions $\vee$, and atomic propositions $I \cup O \cup \{\tau, final\}$, indicating for a state whether certain outgoing edges are present and whether the state is final.

To determine whether a service R is a controller of S, we analyze whether R matches with $OG(S)$, denoted by $R \in Match(OG(S))$. Service R *matches* with $OG(S)$ if R has the same interface as $mp(S)$ and $mp(S)$ simulates R such that, for each pair of simulated states (q_R, q_m) the Boolean formula $\phi(q_m)$ at state q_m of $mp(S)$ is satisfied in the assignment β of state q_R of R. An assignment β is a Boolean function on $I \cup O \cup \{\tau, final\}$ that assigns x, for $x \in I \cup O \cup \{\tau\}$, to true if there exists an outgoing transition from q_R with labels x, and assigns *final* to true if q_R is a final state of R.

Proposition 1 ([4]). *$Controllers(S) = Match(OG(S))$.*

In addition to the operating guidelines of service S, the set of all controllers of a service S can also be represented by a single controller called *maximal controller* (or called *maximal strategy* in [8]) of S, denoted by $M(S)$. For each controlllable service S, every controller R of S can substitute $M(S)$ under accordance [5]. A maximal partner $M(S)$ for S has several useful applications to service substitution [6, 8] and one of them is to characterize the set $Accord(S)$ of all services that can substitute a service S under accordance.

To represent the set $Accord(S)$ of all services that can substitute a service S under accordance, we calculate the operating guidelines of a maximal controller $M(S)$ of S, denoted by $OG(M(S))$. To determine whether a service T accords with S (i. e., if $T \in Accord(S)$), we analyze whether T matches with $OG(M(S))$ [6, 8], denoted by $R \in Match(OG(M(S)))$.

Proposition 2 ([6, 8]). *$Accord(S) = Match(OG(M(S))$.*

Example 3. $OG(M(S))$ depicted in Fig. 1 represents the set $Accord(S)$ of all services that can substitute S under accordance. T_2 matches with $OG(M(S))$ because $OG(M(S))$ simulates T_2 and every state of T_2 fulfills an assignment of a Boolean formula at the respective state of $OG(M(S))$. However, T_1 does not match with $OG(M(S))$ because T_1 terminates after sending abort message $!a$; this means, the assignment at the state in T_1 after sending abort message $!a$ does not satisfy the formula $?r$ at the respective state of $OG(M(S))$. Therefore, $T_1 \notin Match(OG(M(S))$ and $T_2 \in Match(OG(M(S)))$. We conclude from Proposition 2 that T_2 can substitute S under accordance, but T_1 cannot do so.

3 Filtering Guidelines

Given service S, we propose *filtering guidelines* for S which can describe all services S' each has less or the same traces of external messaging behavior to S, yet can substitute S under accordance.

For a service S, a *trace* of S is a finite or infinite sequence of non-internal messages from the initial state of S. The set $\mathit{Traces}(S)$ denotes the set of all traces of S. A service T *refines* service S if every trace of T can be replayed in S (i. e., $\mathit{Traces}(T) \subseteq \mathit{Traces}(S)$). For the attentive reader, this relation is also known as trace refinement relation. The set $\mathit{Refine}(S)$ denotes the set of all services that refines S.

Example 4. In Fig. 1, $T_2 \in \mathit{Refine}(T_1)$ and $T_1, T_2 \notin \mathit{Refine}(S)$.

Definition 1. *A service T is a* filtered service *of a service S if T refines S and T accords with S. We define the set of all filtered services of S as $\mathit{Filter}(S) = \mathit{Refine}(S) \cap \mathit{Accord}(S)$.*

Observe that the set $\mathit{Filter}(S)$ is never an empty set, as it always includes S.

Our first goal is to construct a finite representation of $\mathit{Filter}(S)$. As suggested by Definition 1, the set $\mathit{Filter}(S)$ is an intersection of the two sets $\mathit{Refine}(S)$ and $\mathit{Accord}(S)$. Technically, if each set can be represented by a Boolean annotated service automaton [4, 9], we can employ the operating guidelines-based technique proposed by [9] to characterize the intersection of two given sets using the product of two Boolean annotated service automata.

For this purpose, we require two main ingredients: (1) a finite representation of $\mathit{Accord}(S)$ and (2) a finite representation of $\mathit{Refine}(S)$.

The first ingredient has been proposed by [6, 8] to compute a finite representation of $\mathit{Accord}(S)$ as the operating guidelines of a distinguished controller of S, called a maximal controller $M(S)$ of S (see also Proposition 2 in Section 2).

For the second ingredient, we require a finite representation of $\mathit{Refine}(S)$. Observe that service automaton S is also a natural candidate for our second ingredient. Nevertheless, we would like to employ the operating guidelines-based techniques from [4, 9] to characterize the intersection of two service sets using the product of Boolean annotated service automata. Therefore, we require both of our ingredients to be represented as Boolean annotated service automata.

To compute a finite representation of $\mathit{Refine}(S)$, we first define a *liberal service* of a service S as the deterministic service automaton $L(S)$ such that $L(S)$ has the same interface as S, the same set of traces as S, and simulates S.

Next, we define the annotation function ψ of $L(S)$ as a mapping ψ from the set of states of $L(S)$ to the Boolean formulas over literals $I_{L(S)} \cup O_{L(S)} \cup \{\tau, \mathit{final}\}$. A formula $\psi(q)$ at each state q of $L(S)$ is a disjunction $\vee$ of atomic propositions $x \in I_{L(S)} \cup O_{L(S)} \cup \{\tau, \mathit{final}\}$ where each atomic proposition indicates that either from state q there is an outgoing edge with label x (in case $x \in I_{L(S)} \cup O_{L(S)} \cup \{\tau\}$), or state q is a final state (in case $x = \mathit{final}$).

The following lemma shows that an annotated liberal service $L(S)^{\psi}$ characterizes the set $\mathit{Refine}(S)$.

Lemma 1. *$Refine(S) = Match(L(S)^{\psi})$.*

Proof. We will prove this lemma in two directions.

$\subseteq$: Suppose $T \in Refine(S)$; i. e., $Traces(T) \subseteq Traces(S)$ holds. Because $L(S)$ simulates S by construction, it follows that $L(S)$ also simulates T. Because the formula $\psi(q)$ at state q in $L(S)$ is the disjunction of all literals $x \in I \cup O \cup \{\tau\}$ for an outgoing transition from q with labels x, and of literal *final* if q is a final state in $L(S)$; the Boolean formula $\psi(q)$ at every state q of $L(S)$ is always satisfied in the assignment β of state q_T in T that is simulated by state q in $L(S)$. Thus, $T \in Match(L(S)^{\psi})$ holds.

$\supseteq$: Suppose $T \in Match(L(S)^{\psi})$; i. e., $L(S)$ simulates T and for each simulated pair (q_T, q_L) the state q_T satisfies the formula $\phi(q_L)$. Consider $\sigma \in Traces(T)$. We have $\sigma \in Traces(L(S))$ following from $L(S)$ simulates T. Because $Traces(S) = Traces(L(S))$ by construction, it follows that $\sigma \in Traces(S)$ holds and $Traces(T) \subseteq Traces(S)$ follows. Thus, $T \in Refine(S)$ holds. □

Example 5. Figure 1 shows two annotated liberal services $L(S)^{\psi}$ of S and $L(T_1)^{\psi}$ of T_1. We can see that $T_2 \in Match(L(T_1)^{\psi})$ but $T_2 \notin Match(L(S)^{\psi})$. We conclude from Lemma 1 that $T_2 \in Refine(T_1)$ but $T_2 \notin Refine(S)$.

Given the two ingredients, we employ the product of Boolean annotated service automata [9] to characterize the intersection of *Refine*(S) and *Accord*(S). The product of the two annotated service automata is an annotated service automaton that characterizes the intersection of all services that match with these two service automata. The product of annotated service automata assumes two input annotated service automata with the same interface. Technically, the product can be derived from the synchronous product of two annotated service automata where each state of the product is annotated by the conjunction $\wedge$ of two formulas that contribute to the synchronizing state.

Proposition 3 ([9]). *Let $OG_{\otimes} = OG(S_1) \otimes OG(S_2)$ be the product of two annotated service automata $OG(S_1)$ and $OG(S_2)$, Then $Match(OG_{\otimes}) = Match(OG(S_1))$ $\cap$ $Match(OG(S_2))$.*

We refer to the product of annotated service automata $L(S)^{\psi}$ and $OG(M(S))$ as *filtering guidelines* $FG(S)$ for S, i. e., $FG(S) = L(S)^{\psi} \otimes OG(M(S))$. Theorem 1 shows that $FG(S)$ describes the set *Filter*(S) of all filtered services of S.

Theorem 1. *$Filter(S) = Match(FG(S))$.*

Proof. $Filter(S) = Refine(S) \cap Accord(S)$, [by definition]
$= Match(L(S)^{\psi}) \cap Match(OG(M(S)))$, [Lemma 1, Proposition 2]
$= Match(L(S)^{\psi} \otimes OG(M(S)))$, [Proposition 3]
$= Match(FG(S))$. [by definition] □

Example 6. Figure 1 shows filtering guidelines $FG(S)$ for S which describes all filtered services of S.

Given two services T and S with the same interface but T cannot substitute S, we generalize our filtering guidelines to describe all services T' that have less or the same traces of external messaging behavior to T, yet T' can substitute S under accordance. To this end, we first define a *filtered* service of T for S.

Definition 2. *A service T' is a* filtered *service of T for S if T' refines T and T' accords with S. We define the set of all filtered services of T for S as $\mathit{Filter}(T,S) = \mathit{Refine}(T) \cap \mathit{Accord}(S)$.*

Clearly, $\mathit{Filter}(T,S)$ is possibly an empty set, this means there exists no filtered service of T for S.

Next, we propose a procedure to decide if the set $\mathit{Filter}(T,S)$ of all filtered services of T for S is empty. In case it is not empty, we provide an artifact that represents the set $\mathit{Filter}(T,S)$.

The result from Theorem 1 suggests that we can employ the product of annotated service automata technique to characterize the intersection of $\mathit{Refine}(T)$ and $\mathit{Accord}(S)$. This means, we compute an annotated liberal service $L(T)^{\psi}$ of T as a finite representation of $\mathit{Refine}(T)$, the operating guidelines $OG(M(S))$ as a finite representation of $\mathit{Accord}(S)$, and then the product of the two service automata $L(T)^{\psi}$ and $OG(M(S))$ as a finite representation of $\mathit{Filter}(T,S)$. The product of the $L(S)^{\psi}$ and $OG(M(S))$ is called *filtering guidelines* $FG(T,S)$ of T for S, i. e., $FG(T,S) = L(T)^{\psi} \otimes OG(M(S))$.

Corollary 1. *$\mathit{Filter}(T,S) = \mathit{Match}(FG(T,S))$.*

To decide if there exists a filtered service of T for S, we check if $FG(T,S)$ is empty. An empty $FG(T,S)$ means that it is not possible to synthesize from T a service that both refines T and accords with S. In case of non-empty $FG(T,S)$, we can synthesize a filtered service T' from $FG(T,S)$ by removing all annotations from $FG(T,S)$. The underlying service automaton of $FG(T,S)$ without annotation represents a *most-liberal* filtered service where all undesirable behaviors have been removed only if it is necessary to do so.

Example 7. Figure 1 shows the filtering guidelines $FG(T_1,S)$ of T_1 for S. We see that T_2 matches with $FG(T_1,S)$; therefore, T_2 is a filtered service of T_1 for S (i. e., $T_2 \in \mathit{Filter}(T_1,S)$). However, T_1 does not match with $FG(T_1,S)$; therefore, T_1 itself is not a filtered service of T_1 for S (i. e., $T_1 \notin \mathit{Filter}(T_1,S)$). Given T_1 that cannot substitute S. We can use $FG(T_1,S)$ as guidelines to remove the sending abort message $!a$ from T_1 as it is not described by $FG(T_1,S)$. By doing so, we can derive T_2 that can substitute S from T_1.

We can also apply all existing techniques for operating guidelines to our filtering guidelines $\mathit{Filter}(S)$ and $\mathit{Filter}(T,S)$. For example, suppose we want to impose additional requirements on the service T' that is described by either $\mathit{Filter}(S)$ or $\mathit{Filter}(T,S)$. Then we can restrict our filtering guidelines to services that satisfy certain behavioral constraints [3], or perform certain activities [9].

In case it is not possible to synthesize a service that refines T and accords with S, we can employ the simulation-based graph edit distance approach from [2] on the finite representation of $Accord(S)$ to compute edit actions that are needed for transforming T into T' that accords with T, possibly by means of adding new messaging behavior into T'. Though the simulation-based graph edit distance approach is applicable only for acyclic and deterministic services.

4 Conclusion and Future Work

We presented the notion of a *filtered service* which has less or the same traces of external messaging behavior as a given service, yet can substitute a given service under the substitution criterion called *accordance*. To describe all filtered services, we proposed a finite representation of all filtered services called *filtering guidelines*. The filtering guidelines realizes the *filter* operation by suggesting all possible construction of a filtered service by removing certain undesirable behaviors that are not described by the guidelines. We ensure that the *filtered service* can substitute a given service under accordance.

The idea of the filtering guidelines for services is related to [3] in the sense that all trace-refined services are expressed as behavioral constraints and all trace-refined services that are not substitutable services can be *filtered out*, yielding a customized operating guideline which represents the set of all filtered services.

It is further work to implement our filtering guidelines and obtain experimental results. We also plan to extend our approach to realize the filter operation for services with different input and output channels by blocking some channels of a given service in addition to its external messaging behavior.

References

1. König, D., Lohmann, N., Moser, S., Stahl, C., Wolf, K.: Extending the compatibility notion for abstract WS-BPEL processes. In: WWW 2008. pp. 785–794 (Apr 2008)
2. Lohmann, N.: Correcting deadlocking service choreographies using a simulation-based graph edit distance. In: BPM 2008. LNCS, vol. 5240, pp. 132–147. Springer (Sep 2008)
3. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral constraints for services. In: BPM 2007. LNCS, vol. 4714, pp. 271–287. Springer (2007)
4. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: ICATPN 2007. LNCS, vol. 4546, pp. 321–341. Springer (2007)
5. Mooij, A.J., Voorhoeve, M.: Proof techniques for adapter generation. In: WS-FM 2008. LNCS, vol. 5387, pp. 207–223. Springer (2009)
6. Mooij, A., Parnjai, J., Stahl, C., Voorhoeve, M.: Constructing substitutable services using operating guidelines and maximal controllers. In: WS-FM 2010. To appear (2010)
7. Papazoglou, M.P.: The challenges of service evolution. In: CAiSE 2008. LNCS, vol. 5074, pp. 1–15. Springer (2008)
8. Parnjai, J., Stahl, C., Wolf, K.: A finite representation of all substitutable services and its applications. In: ZEUS 2009. pp. 8–14. CEUR vol. 438 (Mar 2009)
9. Stahl, C., Massuthe, P., Bretschneider, J.: Deciding Substitutability of Services with Operating Guidelines. ToPNoC II 2(5460), 172–191 (Mar 2009)
10. Wolf, K.: Does my service have partners? ToPNoC 5460(II), 152–171 (Mar 2009), special Issue on Concurrency in Process-Aware Information Systems