Service Discovery from Observed Behavior While Guaranteeing Deadlock Freedom in Collaborations

Richard Müller[1,2], Christian Stahl[2], Wil M.P. van der Aalst[2,3], and Michael Westergaard[2,3]

[1] Institut für Informatik, Humboldt-Universität zu Berlin, Germany
Richard.Mueller@informatik.hu-berlin.de
[2] Department of Mathematics and Computer Science,
Technische Universiteit Eindhoven, The Netherlands
{C.Stahl, W.M.P.v.d.Aalst, M.Westergaard}@tue.nl
[3] National Research University Higher School of Economics, Moscow, 101000, Russia

Abstract. Process discovery techniques can be used to derive a process model from observed example behavior (i.e., an event log). As the observed behavior is inherently incomplete and models may serve different purposes, four competing quality dimensions—fitness, precision, simplicity, and generalization—have to be balanced to produce a process model of high quality.

In this paper, we investigate the discovery of processes that are specified as services. Given a service S and observed behavior of a service P interacting with S, we discover a service model of P. Our algorithm balances the four quality dimensions based on user preferences. Moreover, unlike existing discovery approaches, we guarantees that the composition of S and P is deadlock free. The service discovery technique has been implemented in ProM and experiments using service models of industrial size demonstrate the scalability or our approach.

1 Introduction

Service-oriented design [27] reduces system complexity, and service models are useful to understand the running system, to verify the system's correctness, and to analyze its performance. However, it is often not realistic to assume that there exists a service model. Even if there exists a formal model of the implemented service, it can differ significantly from the actual implementation; The formal model may have been implemented incorrectly, or the implementation may have been changed over time. Fortunately, we can often *observe behavior* recorded in the form of an *event log*. Such event logs may be extracted from databases, message logs, or audit trails. Given an event log, there exist techniques to produce a (service) model. The term *service discovery* [6] or, more general, *process discovery* [3] has been coined for such techniques.

In this paper, we assume a known service model S and an event log L containing observed behavior in the form of message sequences being exchanged between (instances of) the implementation of S and (instances of) its environment (i.e., the services S interacts with) to be given. Our goal is to produce a model of the environment of S. As the event log is inherently incomplete (i.e., not all possible behavior was necessarily observed), there are, in general, infinitely many models of the environment of S. Clearly, some models might be more appropriate than others regarding some user requirements. Therefore, service discovery can be seen as a *search process*, aiming at producing a model of the environment that describes the observed behavior "best".

To judge the discovered model we consider two aspects: *correctness* (internal consistency of model, e.g., no deadlocks) and *quality* (ability to describe the underlying observed process well).

Correctness is motivated by the discovery of sound workflow models in [11], where soundness refers to the ability to always terminate [1]. In our service-oriented setting, it is reasonable to require that S and its environment interact correctly. As a minimal requirement of correct interaction, we assume *deadlock freedom* throughout this paper. We refer to such model of the environment of S as a *partner* of S. Thus, we are interested in discovering a partner of S.

Regarding quality, there exist four quality dimensions for general process models [3]: (1) *fitness* (i.e., the discovered model should allow the behavior seen in the event log), (2) *precision* (i.e., the discovered model should not allow behavior completely unrelated to what was seen in the event log), (3) *generalization* (i.e., the discovered model should generalize the example behavior seen in the event log), and (4) *simplicity* (i.e., the discovered model should be as simple as possible). These quality dimensions compete with each other. For example, to improve the fitness of a model one may end up with a substantially more complex model. A more general model usually means a less precise model. We assume that a user guides the balancing of these four quality dimensions. As a consequence, we aim at *discovering a service model that is a partner of S and, in addition, balances the four quality dimensions guided by user preferences.*

The actual challenge is now to find such a model. As a service S has, in general, infinitely many partners, the search space for service discovery is *infinite*. Therefore, we are using a *genetic algorithm* to find a good but possibly not an optimal model of a partner of S. We have implemented this algorithm. It takes as an input a service model S, an event log, and values for the four quality dimensions. The output of the algorithm is a model of a partner of S that comes close to the specified values of the quality dimensions. We show its applicability using eight service models of industrial size. Moreover, based on the notion of a *finite* representation of all partners of S [16]—referred to as *operating guideline*—we additionally apply an *abstraction* that reduces the search space to a finite one. Although the abstraction only preserves fitness, our experimental results shows that the other quality dimensions do not suffer too much due to this abstraction.

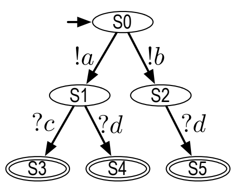
Summarizing, the main contributions of this paper are:

- *adapting existing discovery techniques* for workflows (i.e., closed systems) to services (i.e., reactive systems);
- *adapting the metrics for the four quality dimensions* to cope with service models;
- presenting an approach to *reduce an infinite search space to a finite one*; and
- *validation of the algorithm* based on a prototype.

We continue with a motivating example in Sect. 2. Section 3 provides background information on our formal service model and process discovery techniques. Section 4 adapts existing discovery techniques and metrics for workflows for service mining and reduces the infinite search space to a finite one. We present experimental results in Sect. 5. Section 6 reviews related work, and Section 7 concludes the paper.

2 Motivating Example

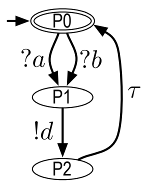
Figure 1 shows a service S modeled as a state machine and an event log L. A transition label $!x$ ($?x$) denotes the sending (receiving) of a message x to (from) the environment of S. The event log L contains information on 210 traces. There are three types of traces: *ac* (10 times), *ad* (100 times), and *bd* (100 times). Our goal is to produce a model of the environment of S. Two example models are P and R in Fig. 1. P incorporates the frequently observed behavior in L (traces *ad* and *bd*) and disregards trace *ac*, arguing that *ac* is negligible for a “good” model. R incorporates even more than the observed behavior in L—for example, the trace *bc* which was not observed in the interaction with S—generalizing the observed behavior in L in account for L’s incompleteness.



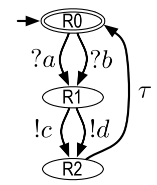
(a) Service S

#	trace
10	*ac*
100	*ad*
100	*bd*
210	

(b) Event log L



(c) Service P



(d) Service R

Fig. 1: Running example: The event log L represents observed communication behavior of S and its environment.

The service P is a partner of S—they both interact without running into a deadlock—whereas the service R is not: If S sends a message b, then R receives this message b and may send a message c. However, S cannot receive message c and R does not send any additional messages unless it receives a message a or b. Thus, the interaction of S and R deadlocks. For this reason, we prefer P over R and our discovery algorithm would exclude R. Classical process mining approaches do not take S into account and will allow for models that deadlock when composed.

3 Preliminaries

For two sets A and B, $A \uplus B$ denotes the disjoint union, i.e., writing $A \uplus B$ expresses the implicit assumption that A and B are disjoint. Let $\mathbb{N}^+$ denote the positive integers. For a set A, $|A|$ denotes the cardinality of A, $\mathcal{B}(A)$ the set of all multisets (bags) over A, and $[\,]$ the empty multiset. Throughout the paper, we assume a finite set of *actions* $\mathcal{A}$ such that $\{\tau, \mathit{final}\} \cap \mathcal{A} = \emptyset$.

For a set A, let A^* be the set of finite sequences (words) over A. For two words v and w, $v \sqsubseteq w$ denotes that v is a *prefix* of w. For a ternary relation $R \subseteq A \times B \times A$, we shall use $a \xrightarrow{b}_R a'$ to denote $(a, b, a') \in R$. If any of the elements a, b, or a' is omitted, we mean the existence of such an element. The relation $R^* \subseteq A \times B^* \times A$ is the *reflexive and transitive closure* of R, defined by $a \xrightarrow{b_1 \ldots b_n}_{R^*} a'$ if and only if there are $a_0, \ldots, a_n \in A$ such that $a = a_0$, $a' = a_n$, and, for all $1 \leq i \leq n$, $a_{i-1} \xrightarrow{b_i}_R a_i$. If $a \longrightarrow_{R^*} a'$, then a' is *reachable* from a in R.

3.1 State Machines for Modeling Services

We model a service as a *state machine* extended by an *interface*, thereby restricting ourselves to the service's communication protocol. An interface consists of two disjoint sets of input and output labels corresponding to asynchronous message channels. In the model, we abstract from data and identify each message by the label of its message channel.

Definition 1 (State Machine). A *state machine* $S = (Q, \alpha, \Omega, \delta, I, O)$ consists of a finite set Q of *states*, an *initial state* $\alpha \in Q$, a set of *final states* $\Omega \subseteq Q$, a *transition relation* $\delta \subseteq Q \times (I \uplus O \uplus \{\tau\}) \times Q$, and two disjoint, finite sets of *input labels* $I \subseteq \mathcal{A}$ and *output labels* $O \subseteq \mathcal{A}$.

Let $l(t) = a$ define the label of a transition $t = (q, a, q') \in \delta$. We canonically extend l to sequences of transitions. For a state $q \in Q$, define by $en(q) = \{a \mid q \xrightarrow{a}_\delta\}$ the set of labels of outgoing transitions of q. The set $\mathcal{R}(S) = \{q \mid \alpha \longrightarrow_{\delta^*} q\}$ denotes the *reachable states* of S. The state machine S is *deterministic* if for all $q, q', q'' \in Q$ and $a \in I \uplus O$, $(q, \tau, q') \in \delta$ implies $q = q'$ and $(q, a, q'), (q, a, q'') \in \delta$ implies $q' = q''$; it is *deadlock free* if, for all $q \in \mathcal{R}(S)$, $en(q) = \emptyset$ implies $q \in \Omega$. ⌟

Graphically, we precede each transition label x with ? (!) to denote an input (output) label. A final state is depicted with a double circle (e.g., *S3* in Fig. 1(a)). An incoming arc denotes the initial state (e.g., *S0* in Fig. 1(a)).

For the composition of state machines, we assume that their interfaces completely overlap. We refer to state machines that fulfill this property as *composable*. We compose two composable state machines S and R by building a product automaton $S \oplus R$, thereby turning all transitions into (internal) τ-transitions. In addition, a multiset stores the pending messages between S and R.

Definition 2 (Composition). Two state machines S and R are *composable* if $I_S = O_R$ and $O_S = I_R$. The *composition* of two composable state machines S and R is the state machine $S \oplus R = (Q, \alpha, \Omega, \delta, \emptyset, \emptyset)$ with $Q = Q_S \times Q_R \times \mathcal{B}(I_S \uplus I_R)$, $\alpha = (\alpha_S, \alpha_R, [\,])$, $\Omega = \Omega_S \times \Omega_R \times \{[\,]\}$, δ containing exactly the following elements:

- $(q_S, q_R, B) \xrightarrow{\tau}_\delta (q'_S, q_R, B)$, if $q_S \xrightarrow{\tau}_{\delta_S} q'_S$,
- $(q_S, q_R, B) \xrightarrow{\tau}_\delta (q_S, q'_R, B)$, if $q_R \xrightarrow{\tau}_{\delta_R} q'_R$,
- $(q_S, q_R, B + [a]) \xrightarrow{\tau}_\delta (q'_S, q_R, B)$, if $q_S \xrightarrow{a}_{\delta_S} q'_S$ and $a \in I_S$,
- $(q_S, q_R, B + [a]) \xrightarrow{\tau}_\delta (q_S, q'_R, B)$, if $q_R \xrightarrow{a}_{\delta_R} q'_R$ and $a \in I_R$,
- $(q_S, q_R, B) \xrightarrow{\tau}_\delta (q'_S, q_R, B + [a])$, if $q_S \xrightarrow{a}_{\delta_S} q'_S$ and $a \in O_S$, and
- $(q_S, q_R, B) \xrightarrow{\tau}_\delta (q_S, q'_R, B + [a])$, if $q_R \xrightarrow{a}_{\delta_R} q'_R$ and $a \in O_R$. ⌟

We compare two state machines S and R by a *simulation relation*, thereby treating τ like any action in $\mathcal{A}$. A binary relation $\varrho \subseteq Q_S \times Q_R$ is a *simulation relation* of S by R if (1) $(\alpha_S, \alpha_R) \in \varrho$, and (2) for all $(q_S, q_R) \in \varrho$, $a \in \mathcal{A} \uplus \{\tau\}$, $q'_S \in Q_S$ such that $q_S \xrightarrow{a}_S q'_S$, there exists a state $q'_R \in Q_R$ such that $q_R \xrightarrow{a}_R q'_R$ and $(q'_S, q'_R) \in \varrho$. If such a ϱ exists, we say that R *simulates* S. A simulation relation ϱ of S by R is *minimal*, if for all simulation relations ϱ' of S by R, $\varrho \subseteq \varrho'$.

We want the composition of two services to be *correct*. As a minimal criterion for correctness, we require deadlock freedom and that every reachable state contains only finitely many pending messages (i.e., the message channels are bounded). We refer to services that interact correctly as *partners*.

Definition 3 (b-Partner). Let $b \in \mathbb{N}^+$. A state machine R is a *b-partner* of a state machine S if $S \oplus R$ is deadlock free and for all $(q_S, q_R, B) \in \mathcal{R}(S \oplus R)$ and all $a \in I_S \uplus I_R$, $B(a) \leq b$. ⌟

In Fig. 1, P is a 1-partner of S, but R is not because the composition $S \oplus R$ can deadlock.

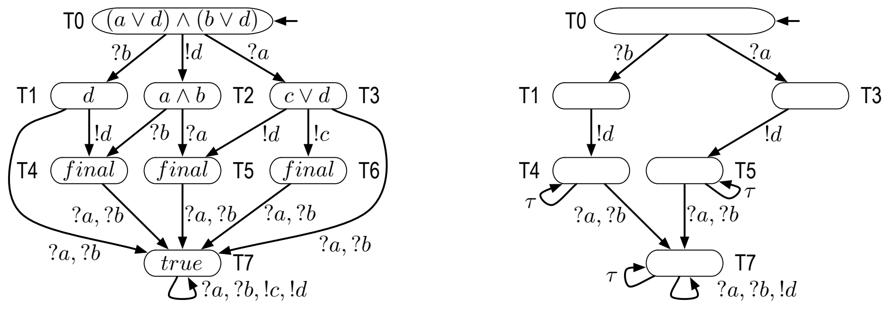
If a state machine S has one b-partner, then it has infinitely many b-partners. Lohmann et al. [16] introduce operating guidelines as a way to represent the infinite set of b-partners of S in a finite manner. Technically, an operating guideline is a deterministic state machine T where each state is annotated with a Boolean formula Φ, which specifies the allowed combinations of outgoing transitions. A state machine R is represented by an operating guideline if (1) there exists a minimal simulation relation ϱ of R by T (as T is deterministic, ϱ is uniquely defined); and (2) for every pair of states $(q_R, q_T) \in \varrho$, the outgoing transitions of q_R and the fact whether q_R is a final state must define a satisfying assignment to $\Phi(q_T)$.

Definition 4 (b-Operating Guideline). An *annotated state machine* (T, Φ) consists of a deterministic state machine T and a Boolean annotation Φ, assigning to each state $q \in Q$ of T a *Boolean formula* $\Phi(q)$ over the literals $I \uplus O \uplus \{\tau, \mathit{final}\}$.

A state machine R *matches* with (T, Φ) if there exists a minimal simulation relation ϱ of R by T such that for all $(q_R, q_T) \in \varrho$, $\Phi(q_T)$ evaluates to *true* for the following assignment β: $\beta(a) = \mathit{true}$ if $a \neq \mathit{final} \wedge q_R \xrightarrow{a}_{\delta_R}$ or $a = \mathit{final} \wedge q_R \in \Omega_R$, and $\beta(a) = \mathit{false}$ otherwise.

Let $b \in \mathbb{N}^+$. The *b-operating guideline* $OG_b(S)$ of a state machine S is an annotated state machine such that for all state machines R, R matches with $OG_b(S)$ iff R is a b-partner of S. ⌟

Figure 2a depicts $OG_1(S) = (T, \Phi)$ of the service S. The state machine P (Fig. 1c) matches with (T, Φ): The minimal simulation relation of P by T is $\varrho = \{(P_0, T_0),$ $(P_1, T_3), (P_1, T_1), (P_2, T_5), (P_2, T_4), (P_0, T_5), (P_0, T_4), (P_1, T_7), (P_2, T_7), (P_0, T_7)\}$, and the formula Φ is evaluated to true, for all pairs of ϱ. For example, for (P_0, T_0) we have $\Phi(P_0) = (\mathit{true} \vee \mathit{false}) \wedge (\mathit{true} \vee \mathit{false})$ which is true and for (P_0, T_4) we have $\Phi(T_4) = \mathit{true}$. Thus, P is a 1-partner of S. Figure 2b depicts the smallest subgraph G of $OG_1(S)$ such that P is still simulated by G, i.e., the subgraph used for the simulation relation above. In contrast, the state machine R (Fig. 1d) does not match with (T, Φ), because (R_1, T_1) violates the simulation relation: We have $R_1 \xrightarrow{!c}$ but $T_1 \not\xrightarrow{!c}$. Thus, R is not a 1-partner of S.



(a) $OG_1(S)$. (b) Subgraph G of $OG_1(S)$.

Fig. 2: $OG_1(S)$ and its smallest subgraph G such that P is simulated by G. The annotation of a state is depicted inside the state. For $OG_1(S)$, every state has a τ-labeled self-loop and the annotation an additional disjunct τ, which is omitted in the figure for reasons of readability.

In the remainder of the paper, we abstract from the actual bound chosen and use the terms partner and operating guideline rather than b-partner and b-operating guideline.

3.2 Event Logs and Alignments

An *event log* is a multiset of traces. Each trace describes the communication between S and R in a particular case in terms of a sequence of events (i.e., sent and received messages). We describe an event as an action label and abstract from extra information, such as the message content or the timestamp of the message. Formally, a trace $w \in \mathcal{A}^*$ is a sequence of actions, and $L \in \mathcal{B}(\mathcal{A}^*)$ is an event log.

To compare a (discovered) service model R with the given event log L, we use the alignment-based approach described in [4]. This approach relates each trace $w \in L$ to a sequence σ of transitions of R that can be executed from R's initial state by pairing events in w to events of σ.

Formally, a *move* is a pair $(x, y) \in \big((\mathcal{A} \uplus \{\gg\}) \times (\delta_R \uplus \{\gg\})\big) \setminus \{(\gg, \gg)\}$. We call (x, y) a *move in the model* if $x = \gg \wedge y \neq \gg$, a *move in the log* if $x \neq \gg \wedge y = \gg$, a *synchronous move* if $x \neq \gg \wedge y \neq \gg$, and a *silent move* if $x = \gg \wedge y \neq \gg \wedge l(y) = \tau$.

An *alignment* of a trace $w \in L$ to R is a sequence $\gamma = (x_1, y_1) \ldots (x_k, y_k)$ of moves, such that the projection of $(x_1 \ldots x_k)$ to $\mathcal{A}$ is w; the projection of $(y_1 \ldots y_k)$ to δ_R is $(\alpha_R, a_1, q_1) \ldots (q_{j-1}, a_j, q_j)$; and transition label $l(y_i)$ and action x_i coincide for every synchronous move (x_i, y_i) of γ. Let $trace(\gamma) \in \mathcal{A}^*$ denote the word by removing all τ-labels from $l(y_1) \ldots l(y_k)$.

Some alignments for L and P in Fig. 1 are:

$$\gamma_1 = \begin{array}{|c|c|} a & c \\ \hline a & \gg \\ (P_0, a, P_1) & \end{array} \quad \gamma_2 = \begin{array}{|c|c|} a & d \\ \hline a & d \\ (P_0, a, P_1) & (P_1, d, P_2) \end{array} \quad \gamma_3 = \begin{array}{|c|c|} b & d \\ \hline b & d \\ (P_0, b, P_1) & (P_1, d, P_2) \end{array}$$

The top row of γ_1 corresponds to the trace $ac \in L$ and the bottom two rows correspond to the service P. There are two bottom rows because multiple transitions of P may have the same label; the upper bottom row consists of transition labels, and the lower bottom row consists of transitions. We have $\alpha_P \xrightarrow{a}_{\delta_P^*}$ but $\alpha_P \not\xrightarrow{ac}_{\delta_P^*}$; that is, ac deviates from a by adding an additional c-labeled transition. Thus, a move in the log, a "$\gg$" appears in the upper bottom row.

The goal is to find a *best alignment* that has as many synchronous and silent moves as possible. The approach in [4] finds such an alignment using a *cost function on moves*. Let γ be an alignment of a trace w to R. Formally, a cost function κ assigns to each move (x, y) of an alignment γ a cost $\kappa((x, y))$ such that a synchronous or silent move has cost 0, and all other types of moves have cost > 0. The cost of γ is $\kappa(\gamma) = \sum_{i=1}^{k} \kappa((x_i, y_i))$; γ is a best alignment if, for all alignments γ' of w to R, $\kappa(\gamma') \geq \kappa(\gamma)$. We use the function λ_R to give us for each trace $w \in L$ a best alignment of w to R.

Finally, we combine the best alignment of each trace of L to R into a weighted automaton AA. A state of AA encodes a sequence of (labels of) transitions of R. We define the weight $\omega(w)$ of each state w as the number of times a trace of L was aligned to w. We shall use AA for the computation of metrics for the two quality dimensions precision and generalization later on.

Definition 5 (Alignment Automaton). The *alignment automaton* $AA(L, R) = (V, v_0, E, \omega)$ of L and R consists of a set of states $V = \mathcal{A}^*$, an initial state $v_0 = \varepsilon$ (ε is the empty trace), a transition relation $E \subseteq V \times \mathcal{A} \times V$ with $v \xrightarrow{a}_E va$ iff there exists $w \in L$ such that $va \sqsubseteq trace(\lambda_R(w))$, and a weight function $\omega : V \to \mathbb{N}^+$ such that $\omega(v) = \sum_{w \in L \wedge v \sqsubseteq trace(\lambda_R(w))} L(w)$ for all $v \in V$. ⌟

Figure 3 depicts the alignment automaton $AA(L, P)$ of the event log L and the state machine P. Each trace in L is either aligned to the transition sequence labeled with a, ad or bd (ignoring τ's), as a transition sequence labeled with ac is not present in P. The weight of each state is depicted inside the state; for example, $\omega(a) = 110$ means 110 traces of L can be aligned to a transition sequence of P whose prefix is a.

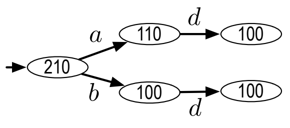


Fig. 3: The alignment automaton $AA(L, P)$

4 Service Discovery

Given a state machine S and an event log L, service discovery aims to produce a service R that is (1) a partner of S and (2) of high quality. The first requirement reduces the search space from all composable services to partners of S and can be achieved by model checking $S \oplus R$ or checking whether R matches with the operating guideline $OG(S)$ of S. In the following, discuss the second requirement.

4.1 Incorporating the Quality Dimensions

We are interested in discovering a partner of highest quality. Numerous metrics for measuring the four quality dimensions have been developed [4,7,29]. However, we cannot simply use these metrics but have to adapt them to cope with service models.

Fitness Let R be a partner of S and L an event log. Fitness indicates how much of the behavior in the event log L is captured by the model R. A state machine with good fitness allows for most of the behavior seen in the event log. We redefine the cost-based fitness metrics from [4] for state machines: We quantify fitness as the total alignment costs for L and R (computed using the best alignments provided by λ_R) compared to the worst total alignment costs. The worst total alignment costs are the sum of "log-only moves" for the events in the observed trace and "model-only moves" to reach a final state. For the latter, we consider the "least expensive path" because a best alignment will try to minimize costs [4].

⟨TODO: Check: original formulation seemed wrong.⟩

Definition 6 (Fitness). The *fitness* of L and R is defined by
$$fit(L,R) = 1 - \frac{\sum_{w \in L} \big(L(w) \cdot \kappa(\lambda_R(w))\big)}{\sum_{w \in L} \big(L(w) \cdot \sum_{x \in w} \kappa((x, \gg))\big)}.$$ ⌟

Assume a cost function κ where each synchronous and each silent move has cost 0, and all other types of moves have cost 1. The best alignments given by λ_P are γ_1–γ_3. We have costs of 1 for γ_1, 0 for γ_2, and 0 for γ_3; therefore, we calculate $fit(L,P) = 1 - \frac{10 \cdot 1 + 100 \cdot 0 + 100 \cdot 0}{10 \cdot 2 + 100 \cdot 2 + 100 \cdot 2} \approx 0.976$. As expected, the fitness value is high because only 10 out of 210 traces are nonfitting traces in L (i.e., the traces ac).

Simplicity Simplicity refers to state machines minimal in structure, which clearly reflect the log's behavior. This dimension is related to Occam's Razor, which states that "one should not increase, beyond what is necessary, the number of entities required to explain anything." Various techniques exist to quantify model complexity [19]. We define the complexity of the model R by its number of states and transitions, and compare it with the smallest subgraph G of $OG(S)$ such that R is simulated by G. Although both R and G have the same behavior, G is not necessarily less complex than R. Our metric takes this into account.

⟨TODO: This part remains a bit weak, but the paper has enough content.⟩

Definition 7 (Simplicity). Let $OG(S) = (T, \Phi)$. The *simplicity* $sim(L,R)$ of L and R is $\frac{|Q_G|+|\delta_G|}{|Q_R|+|\delta_R|}$ if $|Q_G| + |\delta_G| <= |Q_R| + |\delta_R|$ and 1 otherwise, where G is the smallest subgraph of T such that G simulates R. ⌟

Figure 2b shows the smallest subgraph G of $OG(S)$ such that G simulates P. G consists of 6 states and 14 transitions (including the τ-loops at states T_4, T_5, and T_7). Therefore, $|Q_G|+|\delta_G| = 6+14 = 20$ and $|Q_P|+|\delta_P| = 3+4 = 7$; thus, $sim(L,P) = 1$. As expected, L and P have a perfect simplicity value, as P is less complex than G.

Precision Precision indicates whether a state machine is not general. To avoid "underfitting", we prefer state machines with minimal behavior to represent the behavior observed in the event log as closely as possible. We redefine the alignment-based precision metric from [7] for state machines. This metric relies on building the alignment automaton AA, which relates executed and available actions after an aligned trace of the log.

Definition 8 (Precision). Let $AA(L,R) = (V, v_0, E, \omega)$ be the alignment automaton of L and R. Then the *precision* of L and R is defined by $pre(L,R) = \big(\sum_{v\in V}(\omega(v)\cdot|exec(v)|)\big)/\big(\sum_{v\in V}(\omega(v)\cdot|avail(v)|)\big)$, where $exec(v) = en(v)$ in $AA(L,R)$, and $avail(v) = \bigcup_{q\in X} en(q)$ with $X = \{q \mid \alpha_R \xrightarrow{w}_{\delta_R^*} q \wedge w_{|\mathcal{A}} = v\}$ in R. ⌟

Figure 3 shows the alignment automaton $AA(L,P)$, which has been build from the best alignments γ_1–γ_3. We obtain $pre(L,P) = \frac{210\cdot2+110\cdot1+100\cdot1}{210\cdot2+110\cdot1+110\cdot2+100\cdot1+100\cdot2} = 0.6$. As expected, L and P have average precision, as P allows for far more behavior than the behavior observed in L.

⟨TODO: Also add parts $100\cdot0+100\cdot0$?⟩

Generalization Generalization penalizes overly precise state machines which "overfit" the given log. In general, a state machine should not restrict behavior to just the behavior observed in the event log. Often only a fraction of the possible behavior has been observed, e.g., due to concurrency. For this dimension, we developed a new metric. We combine the generalization metric from [4] with the alignment automaton $AA(L,R)$. The idea is to use the estimated probability $\pi(x,y)$ that a next visit to a state w *of the alignment automaton* will reveal a new trace not observed before: $x = |en(w)|$ is the number of unique activities observed at leaving state w, and $y = \omega(w)$ is the number of times w was visited by the event log. We employ an estimator for $\pi(x,y)$, which is inspired by [10].

Definition 9 (Generalization). Let $AA(L,R) = (V, v_0, E, \omega)$ be the alignment automaton of L and R. The *generalization* of L and R is defined by $gen(L,R) = 1 - \big(\frac{1}{|V|}\sum_{v\in V}\pi(|en(v)|, \omega(v))\big)$, where π can be approximated [4] by $\pi(x,y) = \frac{x(x+1)}{y(y-1)}$, if $y \geq x+2$, and $\pi(x,y) = 1$, if $y \leq x+1$. ⌟

We obtain $gen(L,P) = 1 - \frac{1}{5}\big(\frac{2\cdot3}{210\cdot209} + \frac{1\cdot2}{110\cdot109} + \frac{1\cdot2}{100\cdot99}\big) \approx 1$. Given the numbers of traces in L, L and P have nearly perfect generalization as expected, because it is unlikely to reveal a new trace not observed before.

Balancing the Quality Dimensions As quality refers to the possibly competing quality dimensions fitness, simplicity, precision and generalization [3], we cannot assume the existence of a partner that has the highest value for every dimension. We rather need to balance these dimensions and, therefore, assume that a user specified his requirements using weights ω_{fit}, ω_{sim}, ω_{pre}, and ω_{gen}. With these four weights, we can actually search for the partner of S that has highest quality.

Definition 10 (Quality). Let $\omega_{all} = \omega_{fit} + \omega_{sim} + \omega_{pre} + \omega_{gen}$. The *quality* of R for L is defined by $quality(L,R) = \frac{\omega_{fit}}{\omega_{all}} fit(L,R) + \frac{\omega_{sim}}{\omega_{all}} sim(L,R) + \frac{\omega_{pre}}{\omega_{all}} pre(L,R) + \frac{\omega_{gen}}{\omega_{all}} gen(L,R)$ ⌟

Using weights of 2 for fitness, precision, and generalization, and a weight of 1 for simplicity (incorporating that the discovered service can be simpler than its simulation subgraph), we obtain $quality(L,P) = \frac{2}{7} \cdot 0.976 + \frac{1}{7} \cdot 1 + \frac{2}{7} \cdot 0.6 + \frac{2}{7} \cdot 1 \approx 0.879$.

4.2 A Finite Abstraction of the Search Space

The actual challenge of service discovery is that the search space is the set partners of S, which is infinite. In the following we present abstraction that reduces the search space to a *finite* number of partners. To this end, we restrict ourselves to partners of S that are *valid subgraphs* of $OG(S) = (T, \Phi)$, i.e., subgraphs of T whose states are connected and contain the initial state of T and that match with $OG(S)$. As T contains only finitely many states, the number of valid subgraphs of $OG(S)$ is finite too. So, instead of investigating any partner of S, we only consider valid subgraphs of $OG(S)$.

However, this finite abstraction comes at a price: Although every valid subgraph is a partner of S, we may have excluded partners of S that have a better quality than any valid subgraph. More precisely, it can be shown that this abstraction only preserves fitness. We do not elaborate on this and refer the interested reader to [25]. The experimental results in the next section illustrate the appropriateness of the abstraction.

5 Experimental Results

In this section, we report on experiments with eight service models of industrial size.

5.1 Algorithm and Implementation

Discovering a partner for a given state machine S and an event log L is challenging because the search space is the infinite set of partners of S. Even the finite abstraction of the search space to valid subgraphs (see Sect. 4.2) may still be too large to search for an optimal candidate exhaustively. Thus, we are using a *genetic algorithm* to find a good but possibly not a best partner. Genetic algorithms have been successfully applied for discovering workflow models [18,11]. A genetic algorithm evolves a population of candidate solutions (i.e., the *individuals*) step-wise (i.e., in *generations*) toward better solutions of an optimization problem. In our setting, an individual is a state machine R. The quality of a candidate solution is determined by the quality of R (see Def. 10).

Our algorithm employs the general procedure of genetic algorithms, which is depicted in Fig. 4. It creates children through the operations crossover (i.e., randomly exchanging subgraphs between two given individuals), mutation (i.e., randomly adding or removing a transition or a final state from a given individual), and replacement (i.e., replacing a randomly chosen individual by a new, randomly generated individual). We employ a combination of four different termination criteria: A time and a generation limit (i.e., the evolution stops after a given amount of time or generations), a stagnation limit (i.e., the evolution stops if the quality of the high-quality individual stagnates a given number of generations), and a quality limit (i.e., the evolution stops if the high-quality individual meets a specified threshold).

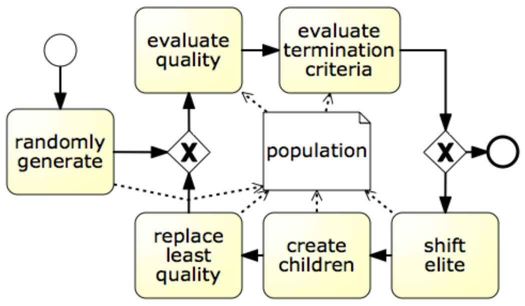


Fig. 4: The different phases of the genetic algorithm

We have implemented the genetic algorithm, both with and without the abstraction presented in Sect. 4.2, in Java as a plug-in [23] in the ProM framework [5].

5.2 Validation

We evaluate the feasibility of our approach by discovering partners for eight service models of industrial size, see Table 1. The services "Loan Approval" and "Purchase Order" are taken from the WS-BPEL specification [14], all other examples are industrial service models provided by a consulting company.

Table 1: Size of the service models, the operating guidelines, and event logs

service S			$OG(S)$		event log L	
name (abbreviation)	$\|Q\|$	$\|\delta\|$	$\|Q\|$	$\|\delta\|$	cases	events
Car Breakdown (CB)	11,381	39,865	1,449	13,863	300	1,938
Deliver Goods (DG)	4,148	13,832	1,377	13,838	300	1,938
Loan Approval (LA)	30	41	21	84	300	2,537
Purchase Order (PO)	402	955	169	1,182	300	2,537
Internal Order (IO)	1,516	4,996	97	567	300	1,938
Ticket Reservation (TR)	304	614	111	731	300	2,381
Reservations (RS)	28	33	370	3,083	300	2,671
Contract Negotiation (CN)	784	1,959	577	4,859	300	1,938

As most services were specified in WS-BPEL, we had to translate them into state machines using the compiler BPEL2oWFN [15]. For each state machine S, we calculated the operating guideline $OG(S)$ using the tool Wendy [17]. Next, we used the underlying state machine T of $OG(S)$ to generate a random event log L using the tool Locretia [13]. Because T is the "most permissive" partner [16] of S, there exists a partner exhibiting the observed behavior in L. Each such event log L is free of noise and consists of 300 cases with about $1,900$–$2,700$ events. Table 1 shows the details. The size of our generated event logs is the size of event logs successfully applied to evaluate the genetic process discovery algorithm in [11]. Finally, we used our implementation to discover a partner of S from $OG(S)$ and L.

As parameters for the genetic algorithm, we used an initial population of 100 individuals, a mutation/crossover/replacement probability of 0.3 with at most 1 crossover point, and elitism of 0.3, i.e., the 30 individuals with the highest quality are directly shifted to the next generation. The algorithm stops after $1,000$ generations, if it stagnates for 750 generations, if a quality of 0.999 is reached, or if it ran for 60 minutes. To take into account that a discovered service can be simpler than the subgraph to be compared, we chose a weight of 1 for simplicity and a weight of 2 for all other dimensions. The experiment is available at [22].

To the best of our knowledge, there does not exist any other service discovery implementation with which we could compare our algorithm. Therefore, we performed two different experiments: one discovering a partner from the complete search space and the other from the abstract search space.

The results in Table 2 show that discovered partners in Experiment 1 are more complex than the ones in Experiment 2; that is, valid subgraphs are smaller than arbitrary partners. This explains the higher computation time in Experiment 1 by a factor of 2–10 compared to Experiment 2: Smaller candidates enable the algorithm to compute more generations in less time.

For the same reason, Experiment 2 produced, in general, partners with higher fitness. The simplicity values are by Def. 6 higher for Experiment 2. However, discovered partners in Experiment 2 have slightly lower precision and generalization values than the partners discovered in Experiment 1. Restricting the search space to valid subgraphs is an abstraction, which neither preserves precision nor generalization. Despite the loss of preservation of the abstraction, the overall quality is in all examples better. In particular, if the discovered partner is in Experiment 1 is too large (e.g., Car Breakdown, Reservations), then the quality of the respective partner discovered in Experiment 2 is much better.

Summing up, our experimental results validate that, in general, partner discovery produces better results on a finite abstraction of the search space than on the complete search space. Although the abstraction only preserves fitness, the values of the other three dimensions and the quality are high.

6 Related Work

The term "service discovery" describes techniques for producing a service model from observed communication behavior of services [6], one the on hand, and techniques for

Table 2: Discovery of an ordinary partner (Experiment 1) and a valid subgraph (Experiment 2) using the genetic algorithm (with *q*uality and *t*ime) conducted on a MacBook Pro, Intel Core i5 CPU with 2.4 GHz and 8 GB of RAM.

	discovered partner in Experiment 1								discovered partner in Experiment 2							
S	$\|Q\|$	$\|\delta\|$	q	*fit*	*sim*	*pre*	*gen*	t	$\|Q\|$	$\|\delta\|$	q	*fit*	*sim*	*pre*	*gen*	t
CB	$> 31k$	$> 31k$	0.58	0.11	0.02	0.9	1	$1h$	75	375	0.96	0.9	1	0.98	0.93	$1h$
DG	569	568	0.61	0.14	0.14	0.92	1	$1h$	225	845	0.97	0.91	1	0.98	0.98	$29m$
LA	33	53	0.91	0.87	0.69	0.99	1	$30m$	14	29	0.98	0.98	1	0.98	0.97	$2m$
PO	125	168	0.92	0.96	0.58	1	0.98	$1h$	81	243	0.98	0.91	1	1	1	$5m$
IO	61	60	0.57	0.2	0.38	0.85	0.75	$1h$	9	12	0.89	0.62	1	0.97	0.96	$51m$
TR	129	219	0.92	0.92	0.64	0.99	0.99	$1h$	28	111	0.97	0.95	1	0.98	0.94	$3m$
RS	$> 24k$	$> 24k$	0.48	0.33	0.02	0.33	1	$1h$	104	372	0.98	1	1	0.91	1	$3m$
CN	721	720	0.59	0.07	0.05	0.98	1	$1h$	40	119	0.93	0.81	1	0.98	0.94	$1h$

finding a service model in a service repository in service-oriented architectures [27], on the other hand. In this paper, we investigated the discovery of a service model from observed communication behavior, which corresponds to a particular form of process mining [3]. Process mining research has been focused on workflows (i.e., closed systems) but during the last few years, process mining techniques have also been applied to services resulting in the term "service mining". Paper [2] reviews service mining research and identifies two main challenges regarding the discovery of services: (1) the correlation of instances of a service with instances of another service (e.g., [9,21]) and (2) the discovery of services based on observed behavior (e.g., [12,28,26,8,30,20]). This paper contributes to the second challenge.

In [24], we considered with weak termination a stronger correctness criterion than deadlock freedom but solely focused on the fitness dimension, thus, ignored the three other quality dimensions. To make the discovery efficient, we do not discover a "best" model as in [24] but a model of high quality using a genetic algorithm. The idea of using an genetic algorithm is inspired by the work of Buijs et al. [11] on discovering sound workflow models while balancing the four conflicting quality dimensions. In Sect. 4, we discussed the relation of our metrics for these four quality dimensions and the metrics used in [11]. For the simplicity metric, we used the structure of the operating guideline, which does not exist for workflow models. Correctness in our setting is deadlock freedom of the service composition, a weaker criterion than soundness in [11]. To deal with correctness in the setting of services, we assume a service S to be given and we discover a partner of S from observed behavior of S.

Musaraj et al. [26] correlate messages from an event log without correlation information and use this information in their discovery algorithm. In contrast, we abstract from correlation information and assume cases to be independent. Furthermore, our approach produces a partner of a given service model S and balances the four conflicting quality dimensions guided by user preferences. Motahari-Nezhad et al. [20] only consider the fitness and the precision dimension and ignore generalization and simplicity of the discovered service. Like Musaraj et al. [26], they do not assume a service model to be given and, thus, they cannot guarantee that their produced service model can interact correctly with its environment. Other approaches discover workflow models from service interaction [12] from interaction patterns [8,30]. Whereas our algorithm produces a complete service model, [12,8,30] can only discover parts of a service.

7 Conclusion and Future Work

We presented a technique to discover a service model from a given service S and observed behavior of a service P interacting with S. Our technique produces a service model for P that can interact correctly (no deadlocks) with S and, in addition, balances the four conflicting quality dimensions (i.e., fitness, simplicity, precision, and generalization). As an additional improvement, we proposed an abstraction technique to reduce the infinite search space to a finite one. As an exhaustive search to find an optimal solution may still be intractable, we implemented our technique as a genetic algorithm. In a prototypical implementation, we experimented with several service models of industrial size. Our results showed that the algorithm finds (nearly) optimal solutions in acceptable time. It is worth mentioning that our approach is not restricted to service models but can discover arbitrary reactive systems.

In future work, we aim to extend our presented approach by improving the simplicity metrics, studying the impact of different weights of the quality dimensions on the quality of the discovered partner, and investigating how the abstraction technique based on valid subgraphs can be improved such that it preserves all metrics. We also plan to extend our approach to stronger correctness criteria than deadlock freedom, e.g., weak termination (i.e., the possibility to always terminate in a service composition).

Acknowledgement Support from the Basic Research Program of the National Research University Higher School of Economics is gratefully acknowledged.

⟨TODO: Change references 22 and 23 into footnotes? Saves space and avoids having too many self references.⟩

References

1. Aalst, W.M.P.v.d.: The application of Petri nets to workflow management. Journal of Circuits, Systems, and Computers 8(1), 21–66 (1998)
2. Aalst, W.M.P.v.d.: Service mining: Using process mining to discover, check, and improve service behavior. IEEE Transactions on Services Computing (2012)
3. Aalst, W.M.P.v.d.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
4. Aalst, W.M.P.v.d., Adriansyah, A., Dongen, B.F.v.: Replaying history on process models for conformance checking and performance analysis. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 2(2), 182–192 (2012)
5. Aalst, W.M.P.v.d., Brand, P.v.d., Dongen, B.F.v., Günther, C.W., Verbeek, E.: ProM 6.2. `http://www.promtools.org/prom6/`
6. Aalst, W.M.P.v.d., et al.: Process mining manifesto. In: BPM 2011 Workshops Proceedings. pp. 169–194. Springer-Verlag (2012)
7. Adriansyah, A., Munoz-Gama, J., Carmona, J., Dongen, B., Aalst, W.: Alignment based precision checking. In: BPI Workshops. lnbip, vol. 132, pp. 137–149. Springer (2013)

8. Asbagh, M., Abolhassani, H.: Web service usage mining: mining for executable sequences. In: WSEAS 2007. vol. 7, pp. 266–271 (2007)
9. Basu, S., Casati, F., Daniel, F.: Toward web service dependency discovery for SOA management. In: SCC 2008. vol. 2, pp. 422 –429 (2008)
10. Boender, C., Rinnooy Kan, A.: A bayesian analysis of the number of cells of a multinomial distribution. The Statistician pp. 240–248 (1983)
11. Buijs, J.C.A.M., Dongen, B.F.v., Aalst, W.M.P.v.d.: On the role of fitness, precision, generalization and simplicity in process discovery. In: CoopIS 2012. LNCS, vol. 7565, pp. 305–322. Springer (2012)
12. Dustdar, S., Gombotz, R.: Discovering web service workflows using web services interaction mining. Int. Journal of Business Process Integration and Management 1(4), 256–266 (2006)
13. Heiden, S., Müller, R.: Locretia - generating logs. `http://svn.gna.org/viewcvs/service-tech/trunk/locretia/`
14. Jordan, D., et al.: Web services business process execution language version 2.0. OASIS Standard 11 (2007)
15. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In: WS-FM 2007. LNCS, vol. 4937, pp. 77–91. Springer-Verlag (2008)
16. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: ICATPN 2007. LNCS, vol. 4546, pp. 321–341. Springer (2007)
17. Lohmann, N., Weinberg, D.: Wendy: A tool to synthesize partners for services. Fundam. Inform. 113(3-4), 295–311 (2011)
18. Medeiros, A., Weijters, A., Aalst, W.M.P.v.d., et al.: Genetic process mining: an experimental evaluation. Data Mining and Knowledge Discovery 14, 245–304 (2007)
19. Mendling, J., Neumann, G., van der Aalst, W.M.P.: Understanding the occurrence of errors in process models based on metrics. In: CoopIS 2007, LNCS, vol. 4803, pp. 113–130. Springer (2007)
20. Motahari-Nezhad, H.R., Saint-Paul, R., Benatallah, B.: Deriving protocol models from imperfect service conversation logs. IEEE Trans. Knowl. Data Eng. 20(12), 1683–1698 (2008)
21. Motahari Nezhad, H.R., Saint-Paul, R., Casati, F., Benatallah, B.: Event correlation for process discovery from web service interaction logs. The VLDB Journal 20(3), 417–444 (Sep 2010)
22. Müller, R.: Data for the evaluation. `https://www2.informatik.hu-berlin.de/top/en/www/mitarbeiter/richard_mueller#`
23. Müller, R.: Service discovery plug-in. `https://svn.win.tue.nl/repos/prom/Packages/ServiceDiscovery/`
24. Müller, R., Aalst, W.M.P.v.d., Stahl, C.: Conformance checking of services using the best matching private view. In: WS-FM 2012. LNCS, vol. 7843, pp. 49–68. Springer (2013)
25. Müller, R., Stahl, C., Aalst, W.M.P.v.d., Westergaard, M.: Service discovery from observed behavior while guaranteeing deadlock freedom in collaborations. bpm-center.org (2013)
26. Musaraj, K., Yoshida, T., Daniel, F., Hacid, M.S., Casati, F., Benatallah, B.: Message correlation and web service protocol mining from inaccurate logs. In: ICWS 2010. pp. 259 –266 (2010)
27. Papazoglou, M.: Web Services - Principles and Technology. Prentice Hall (2008)
28. Rouached, M., Gaaloul, W., Aalst, W.M.P.v.d., Bhiri, S., Godart, C.: Web service mining and verification of properties: An approach based on event calculus. In: CoopIS 2006, LNCS, vol. 4275, pp. 408–425. Springer (2006)
29. Rozinat, A., Aalst, W.M.P.v.d.: Conformance checking of processes based on monitoring real behavior. Information Systems 33(1), 64–95 (2008)

30. Tang, R., Zou, Y.: An approach for mining web service composition patterns from execution logs. In: WSE 2010. pp. 53 –62 (2010)