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

Over the past years, there has been a shift from monolithic systems to distributed systems in system development. One prominent computing paradigm that implements this trend is *service-oriented computing* [24]. A service-oriented system is a distributed system that is composed from smaller building blocks called *services*. A service is an autonomous system that has an interface to interact with other services via asynchronous message passing. 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. Nevertheless, most implementations provide some kind of *observed behavior*, commonly referred to as event log [5]. 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* or, more general, *process discovery* has been coined for such techniques [3].

In this paper, we assume a 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".

In this paper, we consider two kinds of user requirements to analyze how good a model describes the observed behavior: *correctness* and *quality*. 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 for the behavior seen in the event log), (2) *precision* (i.e., the discovered model should not allow for 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, as visualized in Fig. 1. 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.*

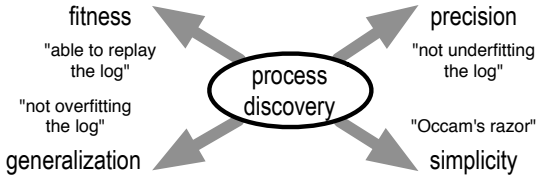


Fig. 1: The different quality dimensions for process model discovery.

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 the 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] called operating guideline, we additionally apply an *abstraction*, ensuring that we only have to check *finitely* many candidates. An operating guideline is a finite state machine, where every state is annotated with a Boolean formula. Our abstraction considers all rooted subgraphs of this state machine rather than all state machines that are simulated by it. Although the abstraction only preserves fitness, we can report on the positive impact in our experimental results.

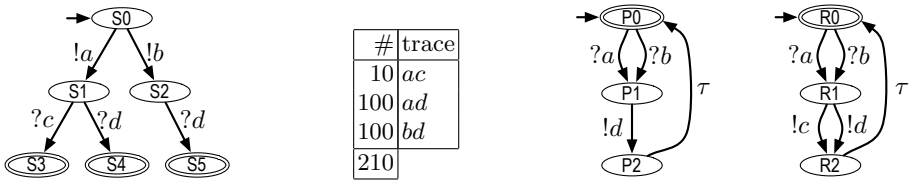
Summing up, we make the following contributions:

- *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 onto services, and Sect. 5 presents an improvement to the discovery process, where we reduce the infinite search space to a finite one. We explain two immediate applications of our approach in Sect. 6 and validate our approach using experimental results on discovering services of industrial size in Sect. 7. Section 8 reviews related work, and Section 9 concludes the paper.

2 Motivating Example

Figure 2 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. 2. They can be justified with L: 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 (b) Event log L (c) Service P (d) Service R

Fig. 2: 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 deadlock freely—whereas the service R is not: If S sends a message b, then R receives this message b and sends a message c. However, S cannot receive message c and R does not send any message 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.

3 Preliminaries

We introduce state machines as a service model and notions, such as event logs and conformance (w.r.t. the quality dimensions) of an event log and a state machine.

3.1 State Machines for Modeling Services

For two sets A and B, $A \uplus B$ denotes the disjoint union; 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.

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 (countable) 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* $O \subseteq \mathcal{A}$.

For a transition $t = (q, a, q') \in \delta$, define its *label* by $l(t) = a$. 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 state machine S is *finite* if the set $\mathcal{R}(S) = \{q \mid \alpha \longrightarrow_{\delta^*} q\}$ of *reachable states* is finite; it 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 $q' = q''$. A state machine $G = (Q', \alpha, \Omega', \delta', I, O)$ is a *rooted subgraph* of S if the states in Q' are connected and $Q' \subseteq Q$, $\Omega' \subseteq \Omega$, $\delta' \subseteq \delta$. ⌟

For technical reasons (i.e., the asynchronous environment in Def. 9), we need to define state machines with countable states, but in general we only consider finite state machines.

Graphically, we precede each transition label x with ? and ! to denote an input and an output label, respectively. A final state is depicted with a double circle; see Fig. 2 for examples.

For the composition of state machines, we assume that their interfaces intentionally 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.

Our composition operator $\oplus$ requires the input and output labeles of S and R to match completely. Technically, we implicitly refer to a preorder between services [27], excluding hierarchies or multi-service composition.

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 by a *simulation relation*, thereby treating τ like any action in $\mathcal{A}$.

Definition 3 (Simulation Relation). Let S and R be two state machines. 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, $\varrho \subseteq \varrho'$. ⌟

If R is deterministic, then there exists a unique minimal simulation relation.

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 4 (Deadlock Freedom, *b*-Partner). Let $b \in \mathbb{N}^+$. A state machine S is *deadlock-free* if, for all reachable states $q \in \mathcal{R}(S)$, $en(q) = \emptyset$ implies $q \in \Omega_S$.

A state machine R is a *b-partner* of S if $S \oplus R$ is deadlock-free and for all reachable states $(q_S, q_R, B) \in \mathcal{R}(S \oplus R)$ and all $a \in I_S \uplus I_R$, $B(a) \leq b$. ⌟

In Fig. 2, P is a 1-partner of S, but R is not because the composition $S \oplus R$ can deadlock.

3.2 Operating Guidelines

If a finite 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, finite state machine where each state is annotated with a Boolean formula.

Definition 5 (Annotated State Machine). An *annotated state machine* (T, Φ) consists of a finite, 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 $I \uplus O \uplus \{\tau, \mathit{final}\}$. ⌟

The procedure for checking whether a state machine R is represented by an annotated state machine (T, Φ) consists of two steps. First, there must exist a minimal simulation relation ϱ of R by T. As T is deterministic, ϱ is uniquely defined. Second, 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 $\Phi(q_T)$. Intuitively, the formula Φ specifies the allowed combinations of outgoing transitions.

Definition 6 (Matching). A state machine R *matches* with an annotated state machine (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:

$$\beta(a) = \begin{cases} \mathit{true}, & \text{if } a \neq \mathit{final} \ \wedge \ q_R \xrightarrow{a}_{\delta_R} \\ \mathit{true}, & \text{if } a = \mathit{final} \ \wedge \ q_R \in \Omega_R \\ \mathit{false}, & \text{otherwise.} \end{cases}$$ ⌟

Finally, we give the definition of an operating guideline [16].

Definition 7 (b-Operating Guideline). 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 composable with S, R matches with $OG_b(S)$ iff R is a b-partner of S. ⌟

Figure 3a depicts $OG_1(S) = (T, \Phi)$ of the service S. The state machine P (Fig. 2c) 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 3b 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. 2d) 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.

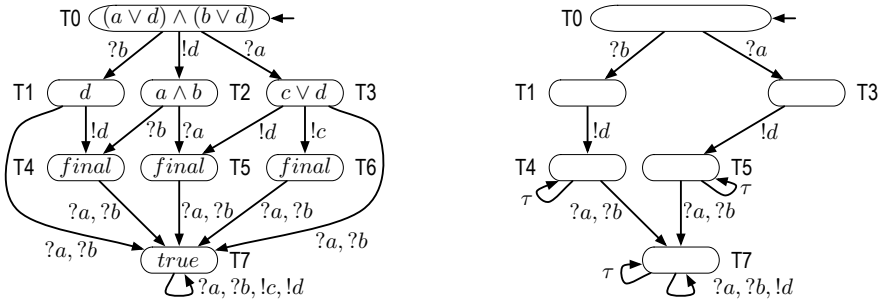
In the remainder of the paper, we abstract from the actual bound chosen and use the terms partner and operating guideline rather than b-partner and b-operating guideline.

3.3 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.

Definition 8 (Event Log). A *trace* $w \in \mathcal{A}^*$ is a sequence of *actions*, and $L \in \mathcal{B}(\mathcal{A}^*)$ is an *event log*. ⌟

An event log may take one out of two *viewpoints* depending on what/when events are recorded in L [22]. If events are recorded when a service R consumes (produces) a message from (for) S, then we can use the *synchronous environment* $env^s(R)$ for checking the conformance of L and R. In contrast, if events are recorded when S consumes (produces) a message from (for) R, then we can use the *asynchronous environment* $env^a(R)$ for comparing L and R. The state machine $env^a(R)$ can be constructed from R by adding to each state of R a multiset containing the pending messages between R and S.



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

Fig. 3: $OG_1(S)$ 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.

Definition 9 (Environment). Let R be a state machine. The *synchronous environment* $env^s(R)$ of R is R. The *asynchronous environment* of R is the state machine $env^a(R) = (Q, \alpha, \Omega, \delta, O_R, I_R)$ defined as

- $Q = Q_R \times \mathcal{B}(I_R \uplus O_R)$,
- $\alpha = (\alpha_R, [\,])$,
- $\Omega = \Omega_R \times \{[\,]\}$,
- δ containing exactly the following elements:
  - $(q_R, B) \xrightarrow{a}_\delta (q_R, B + [a])$, for all $a \in I_R$,
  - $(q_R, B + [a]) \xrightarrow{a}_\delta (q_R, B)$, for all $a \in O_R$,
  - $(q_R, B) \xrightarrow{\tau}_\delta (q'_R, B)$, for all $q_R \xrightarrow{\tau}_{\delta_R} q'_R$,
  - $(q_R, B) \xrightarrow{\tau}_\delta (q'_R, B + [a])$, for all $q_R \xrightarrow{a}_{\delta_R} q'_R$ and $a \in O_R$, and
  - $(q_R, B + [a]) \xrightarrow{\tau}_\delta (q'_R, B)$, for all $q_R \xrightarrow{a}_{\delta_R} q'_R$ and $a \in I_R$. ⌟

The synchronous environment $env^s(P)$ is depicted in Fig. 2c, and Fig. 4 illustrates a part of the asynchronous environment $env^a(P)$. A state machine that is composed with P may send a message a or a message b to P at any time. Therefore, each state of $env^a(P)$ has an outgoing a- and an outgoing b-labeled transition. All internals of P, such as receiving a message a (from state $(P_0, [a])$ to state $(P_1, [\,])$) or sending a message d (from state $(P_1, [\,])$ to state $(P_2, [d])$), are hidden by labeling the respective transitions with τ.

Thus, the choice of the environment of R for comparing L and R depends on what is actually logged in L. In the remainder, we will abstract from these subtle differences and simply write $env(R)$.

We want to compare a (discovered) service model R with the given event log L. For evaluating the quality dimensions of R, we use the approach described in [4] to relate each trace $w \in L$ to a sequence σ of transitions of R that can be executed from R's initial state.

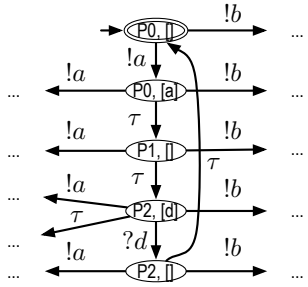


Fig. 4: Illustration of the asynchronous environment $env^a(P)$ of P

Definition 10 (Alignment). Let w be a trace and R be a state machine. A *move* is a pair $(x, y) \in \big((\mathcal{A} \uplus \{\gg\}) \times (\delta_R \uplus \{\gg\})\big) \setminus \{(\gg, \gg)\}$. An *alignment* of w to R is a sequence $\gamma = (x_1, y_1) \dots (x_k, y_k)$ of moves, such that

1. The restriction of γ's first component to $\mathcal{A}$ is the trace w, i.e., $(x_1 \dots x_k)_{|\mathcal{A}} = w$;
2. The restriction of γ's second component to δ_R reaches a state q'_j from the initial state in R, i.e., $(y_1 \dots y_k)_{|\delta_R} = (q_1, a_1, q'_1) \dots (q_j, a_j, q'_j)$ such that $q_1 = \alpha_R$, and for all $1 \leq i < j$, $q'_i = q_{i+1}$;
3. Transition labels and actions coincide (whenever both are defined), i.e., for all $1 \leq i \leq k$, if $x_i \neq \gg$ and $y_i \neq \gg$, then $l(y_i) = x_i$.

A move (x_i, y_i) is a *move in the model* if $x_i = \gg \wedge y_i \neq \gg$, a *move in the log* if $x_i \neq \gg \wedge y_i = \gg$, and a *synchronous move* if $x_i \neq \gg \wedge y_i \neq \gg$. A move (x_i, y_i) such that $x_i = \gg \wedge y_i \neq \gg \wedge l(y_i) = \tau$ is a *silent move*. We denote by $trace(\gamma) \in \mathcal{A}^*$ the word which we derive from the labels of the restriction of γ's second component to δ_R with all τ removed. ⌟

For the trace $ac \in L$ (Fig. 2b) and the state machine P (Fig. 2c), 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, an alignment of ac is $\gamma_1 = (a, a), (c, \gg)$ or graphically

$$\gamma_1 = \left|\begin{array}{c|c} a & c \\ \hline \begin{array}{c} a \\ (P_0, a, P_1) \end{array} & \gg \end{array}\right|$$

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. For a move in the log, a "$\gg$" ("no move") appears in the upper bottom row. For example, in γ_1 the service P cannot perform the last c-action, because c is not the label of an outgoing transition of state P_1. For a move in the model, a "$\gg$" ("no move") appears in the top row.

To choose an alignment that has as many synchronous and silent moves as possible, we use a cost function on moves to find an alignment with the least costs—that is, a "best" alignment.

Definition 11 (Cost Function, Best Alignment). 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, $\kappa(\gamma') \geq \kappa(\gamma)$. ⌟

As there may exist more than one best alignment of w to R, we use an "oracle" function which gives for each trace w of the event log L a best alignment of w to R.

Definition 12 (Oracle Function). Let R be a state machine and let L be an event log. Then λ_R is an *oracle function* if for all $w \in L$, $\lambda_R(w)$ is 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 13 (Alignment Automaton). Let R be a state machine, and let L be an event log. 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 $v_0 = \varepsilon$ corresponds to 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 5 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.

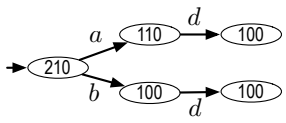


Fig. 5: The alignment automaton $AA(L, P)$

4 Service Discovery

Given a state machine S and an event log L, service discovery aims to produce a partner R of S such that R conforms "best" to L. Therefore, a service discovery algorithm has to guarantee that (1) the discovered service is a partner of S and (2) that the discovered service is of high quality. We address both requirements in the following.

4.1 Task 1: Discovering a Partner

Given a service S, checking whether some service R is a partner of S reduces to either model checking $S \oplus R$ or checking whether R matches with the operating guideline $OG(S)$ of S. As a consequence, the search space for service discovery reduces to all partners of S rather than any service that is composable with S. However, the search space is still infinite. We discuss a further improvement to the discovery process and its restriction in Sect. 5.

4.2 Task 2: Incorporating the Quality Dimensions

In the previous section, we restricted the search space for service discovery to the partners of S. Now, we are interested in a partner of highest quality. 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 a weight for each quality dimension.

To measure the quality of a state machine, we present a metric for each quality dimensions. Numerous metrics for measuring the four quality dimensions have been developed [4,7,26]. In the following, we present our metrics and briefly compare them with the state-of-the-art.

Fitness 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, thereby incorporating the viewpoint of the event log by using the environment of the state machine. We quantify fitness as the total alignment costs for L and $env(R)$ (computed using the optimal alignments provided by the oracle function λ_R) compared to the worst total alignment costs. The worst total alignment costs are just moves in the log and no moves in the model, in all optimal alignments. For the moves in the log only, we consider the "least expensive path" because an optimal alignment will try to minimize costs [4].

Definition 14 (Fitness). The *fitness* of an event log L and a state machine R is defined by

$$fit(L, R) = 1 - \frac{cost(L, env(R))}{move(L)} \quad \text{, where}$$

- $cost(L, env(R)) = \sum_{w \in L} \big(L(w) \cdot \kappa(\lambda_{env(R)}(w))\big)$ are the total alignment costs for L and $env(R)$,
- $move(L) = \sum_{w \in L} \big(L(w) \cdot \sum_{x \in w} \kappa((x, \gg))\big)$ are the total costs of moving through L without ever moving together with $env(R)$. ⌟

Consider P and L in Fig. 2, and assume that L's viewpoint is described by $env(P) = env^s(P) = P$. Assume further 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 the oracle λ_P are γ_1 (see Sect. 3.3) and

$$\gamma_2 = \left|\begin{array}{c|c} a & d \\ \hline a & d \\ (P_0, a, P_1) & (P_1, d, P_2) \end{array}\right| \qquad \gamma_3 = \left|\begin{array}{c|c} b & d \\ \hline b & d \\ (P_0, b, P_1) & (P_1, d, P_2) \end{array}\right|$$

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 Occams Razor, which states that "one should not increase, beyond what is necessary, the number of entities required to explain anything." There exist various techniques to quantify model complexity; see [19] for an overview. We define the complexity of the model by its size, i.e., the number of states and transitions in the underlying graph. Remember that we always discover a state machine R that is a partner of a state machine S. Thus, we measure the difference in size between R and a minimal partner of S with the same behavior as R. A minimal partner of S with the same behavior as R is the smallest subgraph G of $OG(S)$ such that R is simulated by G.

Definition 15 (Simplicity). The *simplicity* of an event log L and a state machine R, which is a partner of a state machine S, is defined by

$$sim(L, R) = \begin{cases} \frac{|Q_G| + |\delta_G|}{|Q_{env(R)}| + |\delta_{env(R)}|}, & \text{if } |Q_G| + |\delta_G| <= |Q_{env(R)}| + |\delta_{env(R)}| \\ 1, & \text{otherwise. ,} \end{cases}$$

where G is the smallest subgraph of $OG(S)$ such that $env(R)$ is simulated by G. ⌟

For our running example, the smallest subgraph G of $OG(S)$ such that P is simulated by G is depicted in Fig. 3b. 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 $|Q_P| + |\delta_P| = 3 + 4 = 7$, and thus $sim(L, P) = 1$. As expected, L and P have a perfect simplicity value, as P is a relatively small state machine compared to the smallest subgraph of G describing L.

Precision Precision indicates whether a state machine is not too 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 16 (Precision). Let R be a state machine, L be an event log, and $AA(L, env(R)) = (V, v_0, E, \omega)$ be the alignment automaton of L and $env(R)$. Then the *precision* of L and R is defined by

$$pre(L, R) = \frac{\sum_{v \in V} \big(\omega(v) \cdot |exec(v)|\big)}{\sum_{v \in V} \big(\omega(v) \cdot |avail(v)|\big)} \quad \text{, where}$$

- $exec(v) = en(v)$ in $AA(L, env(R))$, and
- $avail(v) = \bigcup_{q \in X} en(q)$ with $X = \{q \mid \alpha_{env(R)} \xrightarrow{w}_{\delta^*_{env(R)}} q \wedge w_{|\mathcal{A}} = v\}$. ⌟

For our running example, the alignment automaton $AA(L, env(P))$, which has been build from the best alignments γ_1, γ_2, and γ_3, is depicted in Fig. 5. We obtain $pre(L, P) = \frac{210 \cdot 2 + 110 \cdot 1 + 100 \cdot 0 + 100 \cdot 1 + 100 \cdot 0}{210 \cdot 2 + 110 \cdot 1 + 100 \cdot 2 + 100 \cdot 1 + 100 \cdot 2} = 0.6$. As expected, L and P have average precision, as P allows for far more behavior than the behavior observed in L.

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. For this dimension, we developed a new metric. We combine the generalization metric from [4] with the alignment automaton $AA(L, env(R))$. 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 17 (Generalization). Let R be a state machine, L be an event log, and $AA(L, env(R)) = (V, v_0, E, \omega)$ be the alignment automaton of L and $env(R)$. 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 $y \leq x + 1$. ⌟

For the example, we obtain $gen(L, P) = 1 - \frac{1}{5}\big(\frac{2 \cdot 3}{210 \cdot 209} + \frac{1 \cdot 2}{110 \cdot 109} + \frac{1 \cdot 2}{100 \cdot 99}\big) \approx 1$. 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 To balance these four conflicting quality dimension, we also assume for each of the four quality dimensions a weight ω_{fit}, ω_{sim}, ω_{pre}, and ω_{gen} to be specified by a user. With these four weights, we can actually search for the partner of S that has highest quality.

Definition 18 (Quality). Let L be an event log, R be a state machine, and $\omega_{all} = \omega_{fit} + \omega_{sim} + \omega_{pre} + \omega_{gen}$. 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 equal weights of 1 for each quality dimension, we obtain $quality(L,P) \approx 0.894$ for our running example.

5 Improving the Service Discovery Process

Given a service S, the search space for service discovery is infinite (i.e., all partners of S). For further improving the discovery procedure, we can restrict the search space to a *finite* number of partners. To this end, we consider the operating guideline $OG(S)$ of S and restrict ourselves to partners of S that are *valid subgraphs* of $OG(S)$, i.e., those rooted subgraphs that match with $OG(S)$. As the underlying state machine of $OG(S)$ is finite, the number of rooted subgraphs of $OG(S)$ is finite, too. So instead of investigating any partner of S, we only consider valid subgraphs of $OG(S)$. Subsequently, we discuss the implications of this restriction to valid subgraphs.

Definition 19 (Valid Subgraph). Let (T, Φ) be an annotated state machine. A rooted subgraph G of T is a *valid subgraph* of (T, Φ) if G matches with (T, Φ).⌟

Proposition 20. Any valid subgraph of $OG(S)$ is a partner of S. ⌟

We restricted the search space to the finite set of valid subgraphs of the operating guideline $OG(S)$. This finite abstraction comes at a price: We may have excluded partners of S that have a higher quality than any valid subgraph. Basically, we exclude partners that contain an unfolding of a cycle of $OG(S)$. For an example, consider the partner Q of S in Fig. 6: Rather than having a state with a a-labeled self-loop (state $T7$ from $OG_1(S)$), Q may have a sequence of states (the states $T8$ and $T9$), reached by a sequence of a-labeled transitions (the transitions $(T7, a, T8)$ and $(T8, a, T9)$).

In what follows, we discuss the impact of these "unfolded" partners on the quality of the discovered partners.

Impact on the Fitness Dimension From Def. 14, we conclude that fitness is preserved by the restriction to valid subgraphs. Let R be a partner of S that is not a valid subgraph of $OG(S) = (T, \Phi)$. Let G denote the valid subgraph of $OG(S)$ that is obtained by considering the minimal simulation relation of $env(R)$ by T

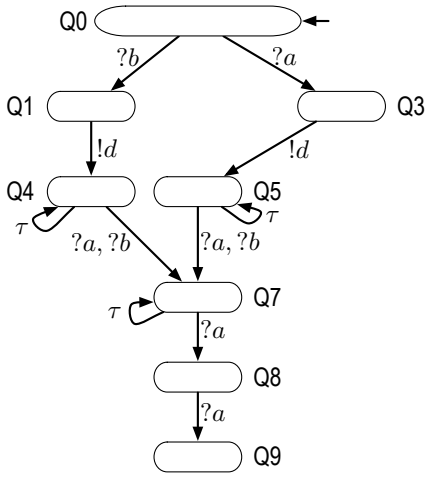
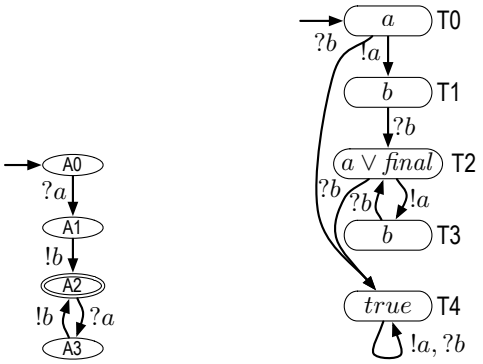


Fig. 6: The service automaton Q, which matches with $OG_1(S)$ but is no valid subgraph.

as a state machine and projecting it onto the states and transitions of T. Clearly, G is uniquely defined. In the following, we conclude that $\mathit{fit}(L, R) \leq \mathit{fit}(L, G)$: Fitness is preserved in G, as $\mathit{env}(R)$ is simulated by G and therefore every trace of $\mathit{env}(R)$ is a trace of G. The presence of τ-transitions does not increase costs, because any τ-transition can only be aligned to "no move"—the resulting silent move always has zero cost by Def. 11.

Impact on the Simplicity Dimension Simplicity from Def. 15 is not preserved by the restriction to valid subgraphs. Consider the services B and G in Figs. 7e and 7d, and assume the viewpoint of log L (Fig. 7c) is the viewpoint of B, i.e., $\mathit{env}(B) = B$. Both B and G are partners of the service A in Fig. 7a. G is a valid subgraph of $OG(A)$ in Fig. 7b, whereas B is not. In addition, G is the smallest subgraph of $OG(A)$ simulating B. Obviously, B is much smaller in size than G. However, they both have a simplicity value of 1: For measuring the simplicity of B, we have to compare the size of B with the size of G according to Def. 15. As $|Q_G| + |\delta_G| = 4 + 4 = 8 > 4 = 2 + 2 = |Q_{\mathit{env}(B)}| + |\delta_{\mathit{env}(B)}|$, we have $\mathit{sim}(L, B) = 1$. On the other hand, as G is a valid subgraph and simulates itself, we have $\mathit{sim}(L, G) = 1$. Therefore, the restriction to valid subgraphs does not preserve simplicity, as we exclude partners from the search space which are obviously simpler, i.e., smaller in size.

Impact on the Precision Dimension Precision from Def. 16 is not preserved by the restriction to valid subgraphs. Consider the service C in Fig. 7f, which is also a partner of service A but no valid subgraph of $OG(A)$. Services G and C have not the same alignment automaton, and C has a higher precision than G:

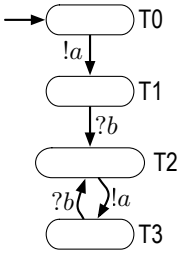


(a) Service A

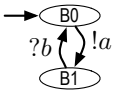
(b) Operating guideline $OG(A)$

#	trace
1	*abab*
1	*ababa*
2	

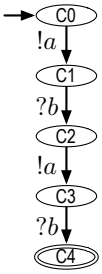
(c) Event log L



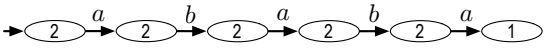
(d) Valid subgraph G of $OG(A)$



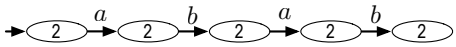
(e) Service B



(f) Service C



(g) Alignment automaton $AA(L, G)$



(h) Alignment automaton $AA(L, C)$

Fig. 7: Technical example for the impact of the restriction on valid subgraphs on the quality.

C has an unfolded cycle of G and thus G has more transitions enabled at the state after trace *abab* of the alignment automaton $AA(L, env(G))$ (see Fig. 7g) than C in its corresponding state (see Fig. 7h).

Impact on the Generalization Dimension Generalization from Def. 17 is not preserved by the restriction to valid subgraphs. The alignment automaton used for generalization is not the same for C and G, which is the sole basis for our generalization metric.

So we can conclude:

Theorem 21. Restricting the search space of service discovery to valid subgraphs preserves fitness. ⌟

6 Applications

We describe two applications of our discovery algorithm: *partner discovery* and *private view discovery*.

The first application of our discovery algorithm is the previously explained partner discovery: Given a service S and an event log L of observed behavior of S interacting with its environment, we can discover a partner P of S such that P and L have, among the abstract partners, the highest quality.

We can also apply our approach for private view discovery [22]. In this setting, we assume a service specification S (the public view) and an event log L to be given. In contrast to partner discovery, L contains observed behavior of the implementation of S (the private view) and its environment. The goal is to produce a private view S' such that S' and L have highest quality. Moreover, every partner of S must also be a partner of S' to ensure that replacing S with S' does not effect any partner of S. Private view discovery has its application in inter-organizational cooperation, where several organizations implement a complex service. To this end, they specify an abstract description of the overall service, the *contract*. Every organization involved implements its share of the contract, which has to conform to the contract. As implementations tend to be very complex, private view discovery is a technique to check whether an implementation is actually correct. As the set of all private views S' of S can be also represented as an annotated state machine, we can use the technique presented in this paper to discover a private view.

7 Experimental Results

In this section, we describe our implementation and evaluate it with our running example and eight service models of industrial size.

7.1 Algorithm and Implementation

Given a state machine S and an event log L recording behavior between a service P and S, we aim to discover a service model of P that is a partner of S and has high quality with L.

As motivated in Sect. 1, the search space (i.e., the number of partners of S) is infinite. Even if we further improve the discovery process by restricting the search space to valid subgraphs (see Sect. 5), it may still be too large to search for an optimal candidate exhaustively. Thus, we are using a *genetic algorithm* to discover a service model of P that has a high but possibly not the maximal quality. Genetic algorithms have been successfully applied in the discovery of

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 either a state machine R (in the discovery process without the improvement from Sect. 5) or a valid subgraph G of $OG(S)$ (in the discovery process with the improvement from Sect. 5). In both cases, the quality of a candidate solution is determined by the quality (see Def. 18) of R and G, respectively, as every valid subgraph is a state machine as well.

Our algorithm employs the general procedure of genetic algorithms, which is depicted in Fig. 8:

1. Choose the initial population (i.e., the first generation) of individuals. These are randomly generated individuals (either state machines matching with $OG(S)$, or valid subgraphs of $OG(S)$). The size of the initial population is part of the input parameters of the algorithm.
2. The algorithm repeats on the current generation until a termination criterion is satisfied:
   (a) Compute the quality of each individual in this generation, using Def. 18.
   (b) *Elitism*: Directly shift a proportion of the individuals with the highest quality into the next generation.
   (c) Select the remaining individuals of the current generation for breeding.
   (d) Create new individuals (called *children*) through crossover, mutation, and replacement operations. The crossover operation randomly exchanges parts (subgraphs) between two given individuals. The mutation operation randomly adds or removes a transition or a final state from a given individual. The replacement operation replaces a randomly chosen individual by a new, randomly generated individual. The probabilities for each operation are part of the input parameters of the algorithm.
   (e) Evaluate the quality of each newly breed individuals.
   (f) Replace the individuals with the least quality in the current generation with high-quality newly breed individuals. They, together with the initially shifted elite individuals, form the new generation.
3. If at least one termination criterion is satisfied, return the individual with the highest quality of the latest generation.

We employ a combination of four different termination criteria to determine when to terminate evolution: (1) A time limit stops the evolution after a certain amount of time, regardless how far the individuals have been evolved. (2) A generation count stops the evolution after a certain number of generations. (3) A stagnation count stops the evolution after a certain number of generations without a new highest-quality individual. (4) A sufficient quality criterion stops the evolution if the quality of the current generation's highest-quality individual exceeds a specified threshold.

We have implemented the genetic algorithm, both with and without the improvement from Sect. 5, in Java as a ProM plug-in[1]. ProM[2]. is an extensible

[1] `https://svn.win.tue.nl/repos/prom/Packages/ServiceDiscovery/`
[2] `http://www.promtools.org/prom6/`

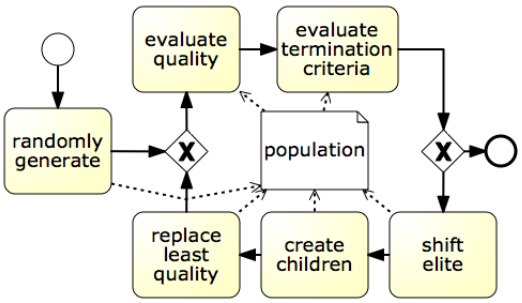


Fig. 8: The different phases of the genetic algorithm.

framework that supports a wide variety of process mining techniques. Our implementation uses the Watchmaker framework[3], which is a framework for implementing platform-independent genetic algorithms in Java. Our implementation completely relies on open source software. The ProM plug-in is free to download and available under the Lesser GNU Public License. ProM is available under the GNU Public License; the Watchmaker Framework is subject to the terms of the Apache Software License, Version 2.0.

7.2 Validation

We evaluate the feasibility of our approach by discovering partners of high quality 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], and the other six 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	$\|Q\|$	$\|\delta\|$	$\|Q\|$	$\|\delta\|$	cases	events
Car Breakdown	11,381	39,865	1,449	13,863	300	1,938
Deliver Goods	4,148	13,832	1,377	13,838	300	1,938
Loan Approval	30	41	21	84	300	2,537
Purchase Order	402	955	169	1,182	300	2,537
Internal Order	1,516	4,996	97	567	300	1,938
Ticket Reservation	304	614	111	731	300	2,381
Reservations	28	33	370	3,083	300	2,671
Contract Negotiation	784	1,959	577	4,859	300	1,938

[3] `http://watchmaker.uncommons.org/`

Figure 9 illustrates our evaluation process. 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]. Columns 4 and 5 of Table 1 depict the size of these operating guidelines. In the next step, we used the underlying state machine T of $OG(S)$ to generate a random event log L. That way, we guarantee that there exists at least one partner exhibiting the observed behavior in L while simultaneously leaving a maximal degree of freedom in generating L, because T is the "most permissive" partner [16] of S. We generated L with the viewpoint of a partner of S (i.e., using env^s) using the tool Locretia[4]. Each such event log L is free of noise and consists of 300 cases with about $1,900$–$2,700$ events—see the last column of Table 1 for the size of the generated event logs. 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 with high quality from $OG(S)$ and L. All experiments were conducted on a MacBook Pro, Intel Core i5 CPU with 2.4 GHz and 8 GB of RAM.

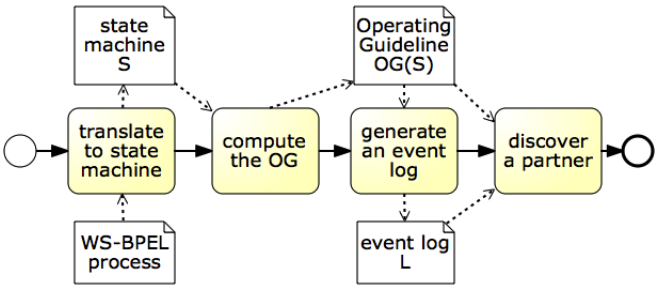


Fig. 9: BPMN diagram illustrating the evaluation process.

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 computation of a new generation stops after $1,000$ generations, if the highest quality stagnates for 750 generations, if a quality of 0.999 is reached, or if the algorithm 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. Notice that because of the restriction to $1,000$ generations, our genetic algorithm generates at most $70,100$ different individuals: 100 individuals for the initial population, and $100 - 30$ individuals (because of elitism) for each generation. Thus, for comparability, we randomly generated

[4] `http://svn.gna.org/viewcvs/service-tech/trunk/locretia/`

71,100 partners in the first experiment, or stopped the random generation after 60 minutes (whatever appeared first). The generated partner with the highest quality is the experiment result. The experiment data is available online[5].

Table 2: Experiment 1: Randomly discover a matching state machine.

	discovered partner							
service name	$\|Q\|$	$\|\delta\|$	*quality*	*fit*	*sim*	*pre*	*gen*	time in *s*
Car Breakdown	210	708	0.64	0.08	0.86	0.84	0.89	3,620
Deliver Goods	62	257	0.7	0.03	1	0.92	1	3,604
Loan Approval	16	15	0.73	0.21	0.71	1	1	3,600
Purchase Order	99	236	0.74	0.54	0.87	0.61	1	3,601
Internal Order	1,490	1,514	0.59	0.14	0.03	0.9	1	3,613
Ticket Reservation	33	98	0.59	0.08	0.98	0.5	1	3,600
Reservations	670	1,426	0.64	0.43	0.61	0.51	1	3,600
Contract Negotiation	117	311	0.57	0.24	0.89	0.69	0.63	3,603
sum								28,841

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 three different experiments on the service models in Table 1. In the first experiment, we randomly generated partners to the given services. We measured their size, their quality, and the time it took to generate them in Table 2. For each service, Table 2 gives the size of the discovered partner (columns 2 and 3), the values of the overall quality and of the individual quality dimensions (columns 4–8), and the time to discover this partner (last column). In the second experiment, we discovered partners that match with the operating guideline, and in the third experiment we discovered partners that are valid subgraphs, as explained in Sect. 5. Tables 3 and 4 show the results.

Based on the experimental results in Tables 2–4, we observe that Experiment 1, which serves as a control group for the two other experiments, has maximal runtime while resulting in lower quality. This shows that our approach is better than guessing. In the remainder, we compare the results of Experiments 2 and 3. The results in Table 3 show that discovered partners in Experiment 2 are more complex than the ones in Experiment 3; that is, valid subgraphs are smaller than arbitrary partners. This explains the higher computation time in Experiment 2 by a factor of 1–44 compared to Experiment 3: Smaller candidates enable the algorithm to compute more generations in less time. For the same reason, Experiment 3 produced, in general, partners with higher fitness. The simplicity values are by Def. 14 higher for Experiment 3. In all examples, the discovered partners in Experiment 3 have slightly higher precision values than the partners discovered in Experiment 2. However, in three out of eight exam-

[5] `https://u.hu-berlin.de/mueller`

ples they have slightly lower generalization values. Restricting the search space to valid subgraphs is an abstraction, which neither preserves precision nor generalization. Therefore, we expected lower precision and generalization values for the partners discovered in Experiment 3, although our experiments confirm only lower generalization values. Despite the loss of preservation of the abstraction, the overall quality of the respective partner discovered in Experiment 3 is in all examples better.

Table 3: Experiment 2: Discover a matching state machine using the genetic algorithm.

			discovered partner					
service name	$\|Q\|$	$\|\delta\|$	*quality*	*fit*	*sim*	*pre*	*gen*	time in s
Car Breakdown	548	1,180	0.72	0.59	0.59	0.7	0.95	3,744
Deliver Goods	246	829	0.71	0.5	0.94	0.57	0.94	3,689
Loan Approval	15	19	0.97	0.91	1	0.98	1	3,239
Purchase Order	101	248	0.94	0.92	0.89	0.99	0.94	3,605
Internal Order	107	106	0.62	0.11	0.19	0.95	1	3,698
Ticket Reservation	29	99	0.91	0.93	1	0.82	0.95	3,606
Reservations	218	671	0.93	0.95	0.98	0.8	1	3,601
Contract Negotiation	73	220	0.62	0.7	1	0.61	0.35	3,798
sum								28,980

Table 4: Experiment 3: Discover a matching state machine using the genetic algorithm with the valid subgraph improvement.

			discovered partner					
service name	$\|Q\|$	$\|\delta\|$	*quality*	*fit*	*sim*	*pre*	*gen*	time in s
Car Breakdown	86	384	0.95	0.87	1	0.99	0.96	3,602
Deliver Goods	82	316	0.96	0.91	1	0.98	0.98	1,763
Loan Approval	14	30	0.98	0.98	1	0.98	0.97	73
Purchase Order	33	107	0.97	0.9	1	1	1	214
Internal Order	9	11	0.87	0.6	1	0.98	0.95	3,021
Ticket Reservation	15	48	0.96	0.96	1	0.99	0.92	143
Reservations	176	582	0.97	1	1	0.91	1	207
Contract Negotiation	74	201	0.94	0.86	1	0.97	0.94	3,606
sum								12,629

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.

8 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 finding a service model in a service repository in service-oriented architectures [24], 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., [13,25,23,8,29,20]). This paper contributes to the second challenge.

In [22], 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 [22] 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.

Dustdar et al. [13] discover workflow models from service interaction. The authors of [8,29] discover workflow models from interaction patterns. These approaches can only discover parts of a service, whereas our algorithm produces a complete service model.

Musaraj et al. [23] 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. Another difference is that our discovered model is a partner of a given service model S (i.e., their composition is deadlock free) and it balances the four conflicting quality dimensions guided by user preferences.

Motahari-Nezhad et al. [20] present a user-driven refinement approach for discovering service models. Their approach considers the fitness and the precision dimension, but ignores generalization and simplicity of the discovered service. Like Musaraj et al. [23], Motahari-Nezhad et al. 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.

9 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. First, we want to change the simplicity metrics such that the size of the candidate is compared with the matching subgraph of the operating guideline that is reduced modulo bisimulation. That way, we can ensure that the candidate cannot be smaller than the respective subgraph. In addition, we will investigate how the abstraction technique based on valid subgraphs can be improved such that it preserves all metric. Second, we aim to consider concurrency in the simplicity metric. To this end, we have to transform the state machine model into a Petri net. However, this comes at a price of drastically increased runtime, even when applying state-of-the-art tools [12]. Third, we plan to extend our approach to the respective precongruence (rather than the preorder) between services [28] and to stronger correctness criteria than deadlock freedom correctness, such as responsiveness (i.e., the possibility to always communicate in a service composition) and weak termination (i.e., the possibility to always terminate in a service composition). Fourth, we plan to study the impact of different weights of the quality dimensions on the quality of the discovered partner.

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

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., Dongen, B.F.v., Herbst, J., Maruster, L., Schimm, G., Weijters, A.J.M.M.: Workflow mining: A survey of issues and approaches. Data Knowledge Engineering pp. 237–267 (2003)
6. Aalst, W.M.P.v.d., et al.: Process mining manifesto. In: BPM 2011 Workshops Proceedings. pp. 169–194. Springer (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. Carmona, J., Cortadella, J., Kishinevsky, M.: Genet: A tool for the synthesis and mining of Petri nets. In: ACSD 2009. pp. 181–185. IEEE (2009)
13. 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)
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 (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 (2010)
22. 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)
23. 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)

24. Papazoglou, M.: Web Services - Principles and Technology. Prentice Hall (2008)
25. 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)
26. Rozinat, A., Aalst, W.M.P.v.d.: Conformance checking of processes based on monitoring real behavior. Information Systems 33(1), 64–95 (2008)
27. Stahl, C., Massuthe, P., Bretschneider, J.: Deciding substitutability of services with operating guidelines. In: ToPNoC II. pp. 172–191. LNCS 5460, Springer (2009)
28. Stahl, C., Vogler, W.: A trace-based service semantics guaranteeing deadlock freedom. Acta Informatica 49(2), 69103 (Feb 2012)
29. Tang, R., Zou, Y.: An approach for mining web service composition patterns from execution logs. In: WSE 2010. pp. 53 –62 (2010)