Conformance Checking of Services Using the Best Matching Private View

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

[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
{W.M.P.v.d.Aalst, C.Stahl}@tue.nl

Abstract. We investigate whether a running implementation of a service conforms to its formal specification in a setting, where only *recorded behavior* of that implementation is given. Existing *conformance checking techniques* can be used to measure the degree of conformance of the recorded behavior and its public view but may produce "false negatives", because a correct implementation (i.e., *private view*) may deviate significantly from its specification. The private view may, for example, reorder some activities without introducing any problems, yet traditional conformance checking would penalize such changes unjustifiably. To overcome this problem, we present a novel approach that determines a *best matching* private view. We show that among the infinitely many private views, there is a *canonical* best matching private view. While the represented theory is general and can be applied to arbitrary service models, the implementation is currently limited to acyclic service models.

1 Introduction

Service-oriented computing (SOC) [17] aims at building complex systems by aggregating less complex, independently-developed building blocks called *services*. A service encapsulates a business functionality and has an interface to interact with its environment—that is, other services—via asynchronous message passing. Aggregating services results again in a service. This modular design of complex systems requires a notion of *service conformance* to safely replace one service (the specification) by another one (the implementation).

Service conformance has been extensively studied in literature (e.g., [6,19]), but most approaches can hardly be used in practice, because they assume that the implementation and the specification of a service are given as *formal models* which do not change over time. However, *it is often not realistic to assume that there exists an up-to-date formal model of the implementation.* Even if there exists a formal model of the implementation, 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 recorded behavior, also referred to as

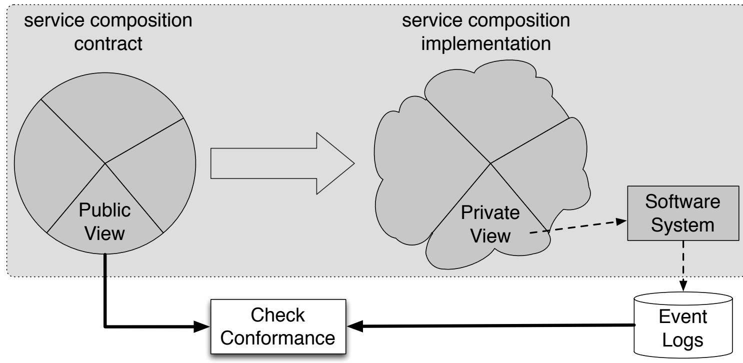


Fig. 1: Illustration of our conformance checking approach

event log, transaction log, or audit trail [3]. Therefore, in this paper, we assume the implementation to be unknown. We only rely on a formal model of the specification and an event log of the unknown implementation. To this end, we answer the question *whether there exists a conforming implementation which may have produced the event log*. Hence, our approach yields a necessary condition for conformance of the unknown implementation with the given specification.

In this paper, we focus on conformance checking based on historic data ("offline" conformance checking). However, the approach can be applied on-the-fly ("online" conformance checking or monitoring); that is, streaming event data can be monitored at runtime and conformance can be checked immediately.

We investigate conformance checking in the setting of a *contract* among *services*. A contract is a (formal) specification of a complex service that involves several cooperating enterprises [1,6]. Later on, each involved party implements its share of the contract. A party's share of the contract—that is, the *public view*—and the implementation thereof may differ significantly but the overall implementation has to conform to the contract. Correctness of a contract (i.e., in our setting, the possibility to always terminate) has been formalized by the *accordance relation* [19]: If every implementation accords with its public view, then the correctness of the contract is preserved and the overall implementation is correct. A party's implementation that accords with the party's public view is a *private view*. Accordance thereby guarantees that any environment that cooperates with a party's public view can cooperate with its respective private view. Instead of checking accordance of the public view and the implementation, we check whether the event log of the implementation conforms to the public view. Figure 1 illustrates a contract involving four parties and its implementation. We use recorded behavior in form of an event log of a running private view to check conformance with its public view.

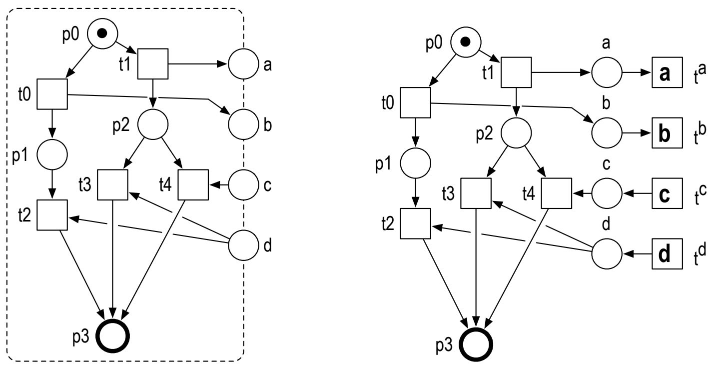
The main contribution of this paper is an approach to check conformance of service in the setting of a contract when the public view of a party's share is a given as a formal model and only observed behavior of its running implementation is known. We show that it is not sufficient to check conformance of the observed behavior with its public view: Accordance allows parties to reorder some activities of their share, but traditional conformance checking would penalize such changes unjustifiably. Therefore, we need to check conformance of the observed behavior with all possible private views instead. However, as there are infinitely many private views in general, this approach is not tractable. We overcome this by proving the existence of a *best matching private view*. If a best matching private view does not conform to L, then no private view does. We present an approach to construct a *canonical* best matching private view from a given public view using existing work on maximal and most-permissive controllers. Moreover, we show how to use a best matching private view not only to check, but to *measure* conformance of an event log with an unknown private view by using existing trace alignment-based techniques from the field of process mining. We have implemented the construction of the canonical best matching private view, yet restricted to acyclic service models, and use the implementation to provide first experimental results.

The remainder is organized as follows. To clarify our setting and our problem statement, we continue with a motivating example in Sect. 2. In Sect. 3, we provide background information on a formal model for services, contracts, and conformance checking. In Sect 4, we show our main result, the existence of a best matching private view. Experimental results on how to compute a canonical best matching private view in Sect. 5 validate our approach. In Sect. 6, we review related work and close with a conclusion.

2 Motivating Example

As a motivating example, consider the public view in Fig. 2a, which is modeled as an open net [21,10]—that is, a Petri net extended with interface places positioned on a dashed frame around the net. The open net *Public* either sends message b and then receives d or sends message a and then receives c or d. A token on place $p3$ models successful termination, also indicated by the thicker bound of place $p3$.

We illustrate the idea of service conformance checking based on observed behavior using open net *Public* and the event log L in Fig. 2c. The event log L thereby represents the recorded behavior of the unknown implementation of *Public*. L contains information of 120 traces, partitioned into three cases. A trace is a sequence of messages sent or received by the implementation. We assume that each event x in a trace of a log corresponds to the sending or receiving of x of the environment of *Public*. We can model this environment of an open net by adding to each x-labeled input place an x-labeled transition that produces tokens on this place and for each x-labeled output place an x-labeled transition that consumes tokens from this place. All other transitions of this environment are internal and, therefore, labeled by τ. Figure 2b illustrates this construction for the public view *Public*; for convenience, we omit all τ labels of transitions. We present a formal definition in Sect. 3.3.



(a) Open net *Public*. (b) Environment of *Public*.

#	trace
30	*ad*
40	*bc*
50	*cda*
120	

(c) Log *L*.

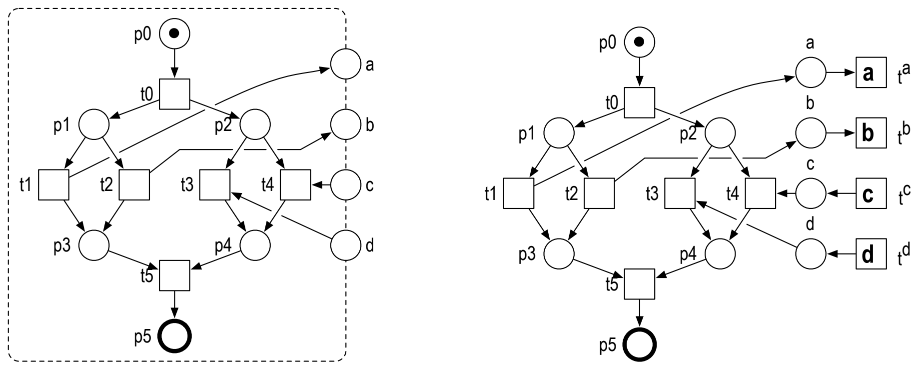
Fig. 2: The public view *Public* and its asynchronous environment $env^a(Public)$. The event log L represents recorded behavior of the implementation of *Public*.

To check whether L conforms to the (environment of the) public view *Public*, we need to replay the traces of L on the model in Fig. 2b. More precisely, we align [2] each trace in L to a trace (i.e., a firing sequence) of the model in Fig. 2b. Some example alignments for L and the environment of *Public* are:

$$\gamma_1 = \left|\begin{array}{c|c|c|c} \gg & a & d & \gg \\ \hline \tau & a & d & \tau \\ t_1 & t^a & t^d & t_3 \end{array}\right| \quad \gamma_2 = \left|\begin{array}{c|c|c|c|c} \gg & b & \gg & \gg & c \\ \hline \tau & b & d & \tau & \gg \\ t_0 & t^b & t^d & t_2 & \end{array}\right| \quad \gamma_3 = \left|\begin{array}{c|c|c|c|c} c & d & \gg & a & \gg \\ \hline c & \gg & \tau & a & \tau \\ t^c & & t_1 & t^a & t_4 \end{array}\right|$$

The top row of each alignment corresponds to "moves in the log" and the bottom two rows correspond to "moves in the model". There are two bottom rows because multiple transitions may have the same label; the upper bottom row consists of transition labels, and the lower bottom row consists of transitions. If a move in the log cannot be mimicked by a move in the model, then a "$\gg$" ("no move") appears in the upper bottom row. For example, in γ_2 the model in Fig. 2b cannot do the last c-move, because c is not connected to the locally enabled transition t_2. If a move in the model cannot be mimicked by a move in the log, then a "$\gg$" ("no move") appears in the top row. For example, all "silent moves" (occurrences of τ-labeled transitions) in the model in Fig. 2b cannot be mimicked by L. Moreover, L did not do a d-move in γ_2 whereas the model in Fig. 2b has to make this move to reach the end. By using this notation, we distinguish between a possible but silent move (depicted by τ) and no move at all (depicted by $\gg$).

Informally, conformance checking of an event log L and a public view N measure "how good" each case in L can be replayed in the environment of N. Thereby, the smaller the number of mismatches in an alignment of a case is, the better this case can be replayed. A mismatch is a move in the log which cannot be mimicked by the model, or a non-silent move in the model which cannot be



(a) Open net *Private*. (b) Environment of *Private*.

Fig. 3: A private view *Private* and its asynchronous environment $env^a(Private)$.

mimicked by the log. Clearly, the more traces we can replay on the model the better the implementation conforms to the public view.

However, even an implementation that accords with its public view may allow for traces that cannot be replayed on the public view, because the accordance relation allows parties to reorder activities of their share, for instance. As an illustration, consider the possible implementation *Private* in Fig. 3a. It is derived from the public view *Public* by parallelizing the sending and receiving of messages. In contrast to the public view, the implementation can, therefore, receive c after having sent b. In this paper, we define correctness of a contract as a finite state-space and the possibility to always terminate. For this definition of correctness, the open net *Private* accords with open net *Public*; that is, the implementation *Private* is a private view of the public view *Public*. Intuitively, every cooperating environment of *Public* knows by receiving either a or b whether *Public* is in the left or the right branch. Therefore, no cooperating environment of *Public* will send c after having received b, as otherwise the cooperation may get stuck. *Public* may operate in such an environment. In fact, it even allows for environments that send c after having received b.

We can replay the event log L on the model of the environment of this open net, which is depicted in Fig. 3b. Some resulting alignments are:

$$\gamma_4 = \begin{array}{|c|c|c|c|c|c|} \gg & \gg & a & d & \gg & \gg \\ \hline \tau & \tau & a & d & \tau & \tau \\ t_0 & t_1 & t^a & t^d & t_3 & t_5 \end{array} \quad \gamma_5 = \begin{array}{|c|c|c|c|c|c|} \gg & \gg & b & c & \gg & \gg \\ \hline \tau & \tau & b & c & \tau & \tau \\ t_0 & t_2 & t^b & t^c & t_4 & t_5 \end{array} \quad \gamma_6 = \begin{array}{|c|c|c|c|c|c|c|} c & d & \gg & \gg & a & \gg & \gg \\ \hline \gg & d & \tau & \tau & a & \tau & \tau \\ & t^d & t_0 & t_1 & t^a & t_3 & t_5 \end{array}$$

Clearly, we can replay more traces on *Private* than on *Public*; that is, the conformance check with the private view gives a better result than the conformance check with the public view. The example clearly shows that, in general, it is not sufficient to check conformance of an event log and the model of the public view. Checking conformance on the public view may generate "false negatives"—that is, acceptable behavior may be diagnosed as non-conforming. As there may exist a private view such that the conformance check with that model gives a better result, we need to check conformance of a log with *all private views*. The challenge thereby is that there exist infinitely many private views. In this paper, we investigate this challenge and present an approach to determine a best matching private view for a given public view.

3 Background

In this section, we provide the basic notions of Petri nets and open nets for modeling services and formalize private view conformance. Suitability of open nets as service model has been demonstrated by feature-complete open net semantics for languages such as BPMN and WS-BPEL [12], and the application of open nets in existing conformance checking techniques [18].

3.1 Petri Nets

As a basic model, we use place/transition Petri nets extended with a set of final markings and transition labels.

Definition 1 (Net). A *net* $N = (P, T, F, m_N, \Omega)$ consists of a finite set P of *places*, a finite set T of *transitions* such that P and T are disjoint, a *flow relation* $F \subseteq (P \times T) \cup (T \times P)$, an *initial marking* m_N, where a marking $m \in \mathcal{B}(P)$ is a multiset over P, and a set Ω of final markings.

A *labeled net* is a net N together with an *alphabet* $\mathcal{A}$ of actions and a *labeling function* $l \in T \to \mathcal{A} \cup \{\tau\}$, where $\tau \notin \mathcal{A}$ represents an invisible, internal action.

Graphically, a circle represents a place, a box represents a transition, and the directed arcs between places and transitions represent the flow relation. A marking is a distribution of tokens over the places. Graphically, a black dot represents a token. We write transition labels beside τ into the respective boxes.

Let $x \in P \cup T$ be a node of a net N. As usual, ${}^\bullet x = \{y \mid (y, x) \in F\}$ denotes the *preset* of x and $x^\bullet = \{y \mid (x, y) \in F\}$ the *postset* of x. We interpret presets and postsets as multisets when used in operations also involving multisets. For markings, we define $+$ and $-$ for the sum and the difference of two markings in the standard way; for example, $[p_1, 2p_2]$ denotes a marking m with $m(p_1) = 1$, $m(p_2) = 2$, and $m(p) = 0$ for $p \in P \setminus \{p_1, p_2\}$. If $m_1 \in \mathcal{B}(P_1)$ and $m_2 \in \mathcal{B}(P_2)$, then $m_1 + m_2 \in \mathcal{B}(P_1 \cup P_2)$ (i.e., the underlying set of elements is adjusted when needed).

The *behavior* of a net N relies on changing the markings of N by firing transitions of N. A transition $t \in T$ is *enabled* at a marking m, denoted by $m \xrightarrow{t}$, if for all $p \in {}^\bullet t$, $m(p) > 0$. If t is enabled at m, it can *fire*, thereby changing the marking m to a marking $m' = m - {}^\bullet t + t^\bullet$. The firing of t is denoted by $m \xrightarrow{t} m'$; that is, t is enabled at m and firing it results in m'.

The behavior of N can be extended to sequences: $m_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{k-1}} m_k$ is a *run* of N if for all $0 < i < k$, $m_i \xrightarrow{t_i} m_{i+1}$. A marking m' is *reachable from* a marking m if there exists a (possibly empty) run $m_1 \xrightarrow{t_1} \ldots \xrightarrow{t_{k-1}} m_k$ with $m = m_1$ and $m' = m_k$; for $w = \langle t_1 \ldots t_{k-1} \rangle$, we also write $m \xrightarrow{w} m'$. Marking m' is *reachable* if it is reachable from initial marking m_N. The set $M_N = \{m' \mid \exists w : m_N \xrightarrow{w} m'\}$ represents all reachable markings of N.

In the case of labeled nets, we lift runs to traces: If $m \xrightarrow{w} m'$ and v is obtained from w by replacing each transition by its label and removing all τ-labels, we write $m \stackrel{v}{\Longrightarrow} m'$. For example, if $w = \langle t_1 t_1 t_2 t_1 t_2 t_3 \rangle$, $l(t_1) = a$, $l(t_2) = \tau$, and $l(t_3) = b$, and $m \xrightarrow{w} m'$, then $m \stackrel{v}{\Longrightarrow} m'$ with $v = \langle aaab \rangle$. The behavior of a labeled net N is described by the runs of N leading from the initial marking to a final marking. The *set of final runs* of a labeled net $N = (P, T, F, m_N, \Omega, l)$ is $R(N) = \{\sigma \in T^* \mid \exists m_f \in \Omega : \ m_N \xrightarrow{\sigma} m_f\}$, and $Tr(N) = \{\sigma \in \mathcal{A}^* \mid \exists m_f \in \Omega : m_N \stackrel{\sigma}{\Longrightarrow} m_f\}$ is the set of *final traces*.

A net N is *bounded* if there exists a bound $b \in \mathbb{N}$ such that for all reachable markings $m \in M_N$ and all places $p \in P$, $m(p) \leq b$. A reachable marking $m \notin \Omega$ of N is a *deadlock* if no transition $t \in T$ of N is enabled at m. If N has no deadlock, then it is deadlock free. A net is *weakly terminating* if from every reachable marking it is always possible to reach a final marking.

3.2 Open Nets

We model services as *open nets* [21,10], thereby restricting ourselves to the communication protocol of a service. In the model, we abstract from data and identify each message by the label of its message channel. An open net extends a net by an *interface*. An interface consists of two disjoint sets of input and output places corresponding to asynchronous input and output channels. An input place has an empty preset, and an output place has an empty postset. In the initial marking and the final markings, interface places are not marked.

Definition 2 (Open net). An *open net* N is a tuple $(P, T, F, m_N, I, O, \Omega)$ with

- $(P \cup I \cup O, T, F, m_N, \Omega)$ is a net such that P, I, O are pairwise disjoint;
- for all $p \in I \cup O$, $m_N(p) = 0$, and for all $m \in \Omega$ and $p \in I \cup O$, $m(p) = 0$;
- the set I of *input places* satisfies for all $p \in I$, ${}^\bullet p = \emptyset$; and
- the set O of *output places* satisfies for all $p \in O$, $p^\bullet = \emptyset$.

Open net N is *sequentially communicating* if each transition is connected to at most one *interface place*. If $I = O = \emptyset$, then N is a *closed net*. Two open nets are *interface-equivalent* if they have the same sets of input and output places.

Graphically, we represent an open net like a net with a dashed frame around it. The interface places are positioned on the frame. If an open net has at most one final marking, we indicate places marked in that final marking with a thicker bound.

For the composition of open nets, we assume that the sets of transitions are pairwise disjoint and that no internal place of an open net is a place of any other open net. In contrast, the interfaces overlap intentionally. We require that all communication is *bilateral* and *directed*; that is, every shared place p has only one open net that sends into p and one open net that receives from p. We refer to open nets that fulfill these properties as *composable*. We compose two composable open nets N_1 and N_2 by merging shared interface places and turn these places into internal places. The definition of composable thereby guarantees that an open net composition is again an open net (possibly a closed net).

Definition 3 (Open net composition). Open nets N_1 and N_2 are *composable* if $(P_1 \cup T_1 \cup I_1 \cup O_1) \cap (P_2 \cup T_2 \cup I_2 \cup O_2) = (I_1 \cap O_2) \cup (I_2 \cap O_1)$. The *composition* of two composable open nets N_1 and N_2 is the open net $N_1 \oplus N_2 = (P, T, F, m_N, \Omega, I, O)$ where

- $P = P_1 \cup P_2 \cup (I_1 \cap O_2) \cup (I_2 \cap O_1)$,
- $T = T_1 \cup T_2$,
- $F = F_1 \cup F_2$,
- $m_N = m_{N_1} + m_{N_2}$,
- $I = (I_1 \cup I_2) \setminus (O_1 \cup O_2)$,
- $O = (O_1 \cup O_2) \setminus (I_1 \cup I_2)$, and
- $\Omega = \{m_1 + m_2 \mid m_1 \in \Omega_1, m_2 \in \Omega_2\}$.

We want the composition of a set of services to be *correct*. Correctness refers to boundedness and weak termination. A user that communicates with a service such that the composition is correct can be seen as a *controller* of this service.

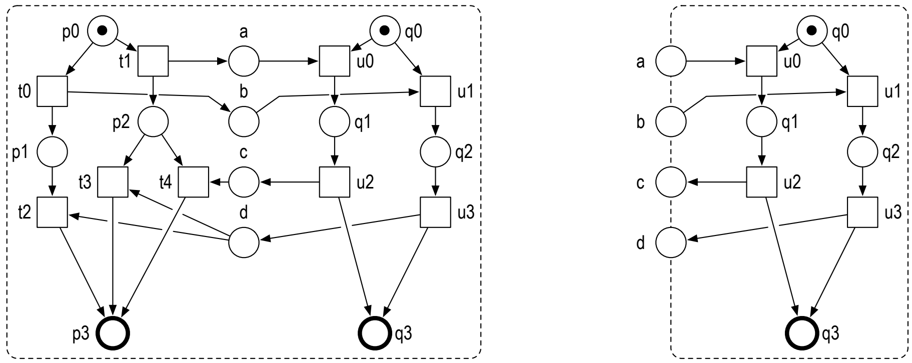
Definition 4 (Controller). Let $b \in \mathbb{N}$. An open net C is a *b-controller* of an open net N if the composition $N \oplus C$ is a closed net, b-bounded, and weakly terminating.

In the remainder of the paper, we abstract from the actual bound chosen and, therefore, use the term controller rather than b-controller for convenience.

Example 1. Consider open nets *Public* in Fig. 2a and *Controller* in Fig. 4b. *Public* has the initial marking $m_{Public} = [p0]$, final markings $\Omega = \{[p3]\}$, output places a and b, and input places c and d. *Controller* has the initial marking $m_{Controller} = [q0]$, final markings $\Omega = \{[q3]\}$, output places c and d, and input places a and b. Clearly, *Public* and *Controller* are composable and their composition *Public* $\oplus$ *Controller* is the closed net in Fig. 4a with initial marking $[p0, q0]$ and final markings $\{[p3, q3]\}$. As *Public* $\oplus$ *Controller* is 1-bounded and weakly terminating, we conclude that *Controller* is a controller of *Public*, and vice versa.

3.3 Private View Conformance

We see a contract as a closed net N, where every transition is assigned to one of the involved parties $X_1, \ldots, X_k$. We impose only one restriction: If a place is



(a) Open net *Public* $\oplus$ *Controller*. (b) Open net *Controller*.

Fig. 4: The open net *Controller* and its composition with the open net *Public*.

accessed by more than one party, it should act as a directed bilateral communication place. This restriction reflects the fact that a party's public view of the contract is a service again. A contract N can be cut into parts $N_1, \dots, N_k$, each representing the agreed public view of a single party X_i ($1 \leq i \leq k$). Hence, we define a contract as the composition of the open nets $N_1, \dots, N_k$.

Definition 5 (Contract). Let $\mathcal{X} = \{X_1, \dots, X_k\}$ be the set of parties and let $\{N_1, \dots, N_k\}$ be a set of pairwise interface-compatible open nets such that $N = N_1 \oplus \dots \oplus N_k$ is a closed net. Then, N is a *contract for* $\mathcal{X}$. For $i = 1, \dots, k$, open net N_i is the *public view of* X_i *in* N and open net $N_i^{-1} = \bigoplus_{j \neq i} N_j$ is the *environment* of X_i in N.

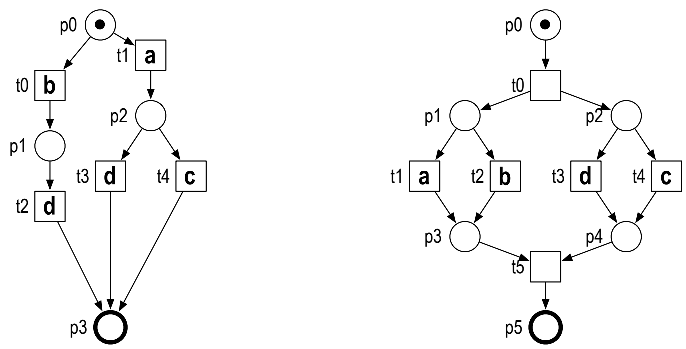
Each Party X_i can independently substitute its public view N_i by a *private view* N_i' if the environment of X_i cannot distinguish between N_i and N_i' [5], which is formalized by the accordance relation [19].

Definition 6 (Accordance). Let N_i and N_i' be interface-equivalent open nets. Open net N_i' *accords with* open net N_i, denoted by $N_i' \sqsubseteq_{acc} N_i$, if every controller of N_i is also a controller of N_i'.

Example 2. An example of a contract involving only two parties is the closed net in Fig. 4a. In Sect. 2, we have motivated that open net *Private* accords with open net *Public*. Thus, we can safely replace *Public* with *Private* without violating the contract.

Sending or receiving a message is an activity. Let $\mathcal{A}$ denote the set of all activities. We define an event log as a multiset of traces over $\mathcal{A}$. Each trace describes the life-cycle of a particular case in terms of the activities executed.

Definition 7 (Event log). An *event log* L_i of the observed behavior of party X_i in contract N is a multiset of traces over $\mathcal{A}$, i.e., $L_i \in \mathcal{B}(\mathcal{A}^*)$.



(a) $env^s(Public)$.

(b) $env^s(Private)$.

Fig. 5: The synchronous environment of open nets *Public* and *Private*. Label τ is omitted.

For conformance checking of party X_i, we compare the observed behavior (event log L_i) with the modeled behavior (N_i or N'_i). We can take two *viewpoints* depending on what/when events are recorded in L_i. If events are recorded when party X_i consumes a message from N_i^{-1} or produces a message for N_i^{-1}, then we can use the *synchronous environment* $env^s(N_i)$ for conformance checking. Here, we label each transition with the adjacent interface places—if possible—and remove the interface places. To simplify the labeling of transitions connected to interface places, we only consider sequentially communicating nets. That way, each transition is labeled by a single label rather than by a set of labels. This restriction is not significant, as every open net can be transformed into an equivalent sequentially communicating open net [10].

Definition 8 (Synchronous environment). The *synchronous environment* of a sequentially communicating open net $N = (P, T, F, m_N, \Omega, I, O)$ is the labeled net $env^s(N) = (P, T, F \cap ((P \times T) \cup (T \times P)), m_N, \Omega, l)$ with $l(t) = p$ where p is the unique interface place $p \in I \cup O$ adjacent to $t \in T$, or $l(t) = \tau$ if no such adjacent interface place exists.

Example 3. Figure 5 shows the synchronous environments of open nets *Public* and *Private*. A transition label is depicted inside a transition with bold font to distinguish it from the transition's identity.

If events are recorded when the environment N_i^{-1} of party X_i consumes a message from party X_i or produces a message for party X_i, then we can use the *asynchronous environment* $env^a(N_i)$ for conformance checking. The net $env^a(N)$ is a net that can be constructed from N by adding to each interface place $p \in I \cup O$ a p-labeled transition t^p in $env^a(N)$. Intuitively, the construction translates the asynchronous interface of N into a synchronous interface with unbounded buffers described by the transition labels of $env^a(N)$.

Definition 9 (Asynchronous environment). The *asynchronous environment* of an open net $N = (P, T, F, m_N, I, O, \Omega)$ is the labeled net $env^a(N) = (P \cup I \cup O, T \cup T', F \cup F', m_N, \Omega, l)$ where

- $T' = \{t^x \mid x \in I \cup O\}$,
- $F' = \{(t^x, x) \mid x \in I\} \cup \{(x, t^x) \mid x \in O\}$, and
- $l(t) = \begin{cases} x, & t^x \in T' \\ \tau, & t \in T. \end{cases}$

Example 4. Figures 2b and 3b show the asynchronous environments of the open nets *Public* and *Private* from Figs. 2a and 3a. A transition label is depicted inside a transition with bold font to distinguish it from the transition's identity.

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

To check conformance, we need to *align* traces in the event log to traces of the service (environment); that is, we need to relate "moves" in the log to "moves" in the model. However, there may be some moves in the log that cannot be mimicked by the model, and vice versa. For convenience, we introduce the set $A_L = \mathcal{A} \cup \{\gg\}$ where $x \in A_L \setminus \{\gg\}$ refers to "move x in the log" and $\gg \in A_L$ refers to "no move in the log". Similarly, for a labeled net N, we introduce the set $A_N = \{(a, t) \in (\mathcal{A} \cup \{\tau\}) \times T \mid l(t) = a\} \cup \{\gg\}$ where $(a, t) \in A_N$ refers to "move a in the model" and $\gg \in A_N$ refers to "no move in the model". A "move τ in the model" (τ, t) is a silent move, as it is only observable by party X_i.

Definition 10 (Alignment). For an event log L and a labeled net N, one *move* in an alignment is represented by a pair $(x, y) \in A_L \times A_N$ such that

- (x, y) is a *move in the log* if $x \in \mathcal{A}$ and $y = \gg$,
- (x, y) is a *move in the model* if $x = \gg$ and $y \in A_N \setminus \{\gg\}$,
- (x, y) is a *move in both* if $x \in \mathcal{A}$ and $y \in A_N \setminus \{\gg\}$,
- (x, y) is an *illegal move* $x = \gg$ and $y = \gg$.

We refer to a move in the model $(x, (a, t))$ with $a = \tau$ as a *silent move.* $A_{LN} = \{(x, y) \in A_L \times A_N \mid x \neq \gg \ \vee y \neq \gg\}$ is the set of all *legal moves.*

An *alignment* of $\sigma \in L$ and $w \in R(N)$ is a sequence $\gamma \in {A_{LN}}^*$ such that the projection on the first element (ignoring $\gg$) yields σ and the projection on the second element (ignoring $\gg$) yields w. The *set of alignments for σ in N* is $\Gamma_{\sigma,N} = \{\gamma \in {A_{LN}}^* \mid \exists w \in R(N) : \gamma \text{ is an alignment of } \sigma \text{ and } w\}$.

Example 5. For an example of an alignment of a trace of an event log and a trace of an open net, consider the six alignments $\gamma_1, \ldots, \gamma_6$ in Sect. 2.

Given a log trace, there may be many possible alignments. To measure the quality of an alignment, we define a *distance function* on legal moves.

Definition 11 (Distance function). A *distance function* $\delta : A_{LN} \to \mathbb{N}$ associates costs to legal moves in an alignment. We define a *standard distance function* δ_S as $\delta_S(a, \gg) = 1$; $\delta_S(\gg, (b, t)) = 1$, for $b \neq \tau$; $\delta_S(\gg, (\tau, t)) = 0$; $\delta_S(a, (b, t)) = 0$, for $a \neq \gg$ and $a = b$; and $\delta_S(a, (b, t)) = \infty$, for $a \neq \gg$ and $a \neq b$.

We generalize a distance function δ to alignments by taking the sum of the costs of all individual moves: $\delta(\gamma) = \sum_{(x,y)\in\gamma} \delta(x, y)$. In δ_S, only moves where log and model agree on the activity, and silent moves of the model have no associated costs. Moves in only the log or model have cost 1, moves where both log and model make a move but disagree on the activity have high costs; thereby, ∞ should be read as a number large enough to discard the alignment. Note that δ_S is just an example cost function; various cost functions can be defined.

Thus far, we considered a *specific* trace of the model. However, our goal is to identify for each log trace the *best matching* trace of the model. Therefore, we define the notion of an *optimal alignment*.

Definition 12 (Optimal alignment). An alignment $\gamma \in \Gamma_{\sigma,N}$ is *optimal* for a log trace $\sigma \in L$ and a labeled net N if for any $\gamma' \in \Gamma_{\sigma,N}$: $\delta(\gamma') \geq \delta(\gamma)$.

If $R(N)$ is not empty, there is at least one (optimal) alignment for any given log trace σ. However, there may be multiple optimal alignments for σ. Since our goal is to align traces in the event log to traces of the model, we nondeterministically select an arbitrary optimal alignment. Therefore, we can construct a function λ_N that provides an "oracle".

Definition 13 (Oracle). Given a log trace σ and a labeled net N, the *oracle* λ_N produces *one* optimal alignment $\lambda_N(\sigma) \in \Gamma_{\sigma,N}$.

The alignments produced by the "oracle" λ_N can be used to quantify conformance of a log L and a model N. Conformance checking involves the interplay of four orthogonal dimensions: *fitness*, *precision*, *generalization*, and *simplicity* [2]. Fitness indicates how much of the behavior in the event log is captured by the model. A model with good fitness allows for most of the behavior seen in the event log. Precision indicates whether the model is not too general. To avoid "underfitting" we prefer models with minimal behavior to represent as closely as possible the behavior seen in the event log. Generalization penalizes overly precise models which "overfit" the given log. In general, a process model should not restrict behavior to just the behavior seen in the event log. Simplicity refers to models 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."

In the remainder, we abstract from the dimensions involved in conformance checking: We assume a function *conf* that computes the conformance of an event log L and a labeled net N based on the alignments produced by the oracle λ_N; that is, $\mathit{conf}(L, N)$ yields a number between 0 (poor conformance) and 1 (perfect conformance) [2]. We define *private view conformance* as the maximal conformance of all private views of a given public view.

Definition 14 (Private view conformance). Let $N = N_1 \oplus \cdots \oplus N_k$ be a contract for $\mathcal{X} = \{X_1, \ldots, X_k\}$. Let N_i be the public view of X_i, and let L_i be an event log of X_i. Let $Pr(N_i) = \{M \mid M \sqsubseteq_{acc} N_i\}$ denote the set of all private views that accord with N_i. Then

– $M \in Pr(N_i)$ is a *best matching private view* for N_i and L_i if for any $M' \in Pr(N_i)$: $conf(L_i, env(M)) \geq conf(L_i, env(M'))$; and
– $conf(L_i, env(M))$ is the *private view conformance* for party X_i where $M \in Pr(N_i)$ is a best matching private view for N_i and L_i.

Definition 14 provides a well-defined conformance notion that can be parameterized with different correctness notions (e.g., deadlock freedom, weak termination) and different environments (e.g., $env^s(N)$, $env^a(N)$). However, Def. 14 cannot easily be transformed into an algorithm. There may be many (if not infinitely many) private views that accord with N_i. So far, no algorithm has been implemented to select a best matching private view. In the next section, we show how private view conformance for party X_i can be decided.

4 Deciding Private View Conformance

In the previous section, we introduced a notion of private view conformance that is independent from the conformance checking dimensions involved. In this section, we decide private view conformance w.r.t. the fitness dimension.

A model with good fitness allows for most of the behavior seen in the event log. Therefore, it is natural to define $conf(L, N)$ inversely proportional to the sum of the costs of aligning all traces of L to traces of N; that is, $conf(L, N)$ should be maximal if $\sum_{\sigma \in L} \delta(\lambda_N(\sigma))$ is minimal. If a trace appears multiple times in the event log, the associated costs should be counted multiple times.

Definition 15 (Fitness). Conformance $conf(L, N)$ w.r.t. *fitness* of an event log L and a labeled net N yields a number between 0 (poor fitness) and 1 (perfect fitness) and is maximal if the *alignment-based costs* $\delta(L, N) = \sum_{\sigma \in L} \delta(\lambda_N(\sigma))$ are minimal.

Our approach for deciding private view conformance does not rely on a specific fitness measure; any fitness measure is suitable as long as it meets the criteria in Def. 15. Our approach relies on the existence of two specific controllers of any open net N: a *maximal controller* $maxC(N)$ [14,8] and a *most permissive controller* $mpC(N)$ [22]. A maximal controller is maximal w.r.t. the accordance relation; that is, every controller of N accords with $maxC(N)$. A most permissive controller $mpC(N)$ is maximal w.r.t. behavior; that is, N can visit all the states in composition with $mpC(N)$ that can be visited in composition with any controller of N. For technical details of maximal and most permissive controllers we refer to [14] and [22], respectively; here, we only summarize their properties.

Proposition 1 ([14]). *For any open net N, there exist controllers $maxC(N)$ and $mpC(N)$ such that for any controller C of N, we have $C \sqsubseteq_{acc} maxC(N)$ and $Tr(env(C)) \subseteq Tr(env(mpC(N)))$.*

Given a contract $N = N_1 \oplus \cdots \oplus N_k$, we show that $B_i = mpC(maxC(N_i))$ is a canonical *best matching private view* for N_i and event log L_i. In other words, open net B_i accords with N_i and has minimal costs and, hence, maximal fitness.

Theorem 2 (Main result). *Let* $N = N_1 \oplus \cdots \oplus N_k$ *be a contract for* $\mathcal{X} = \{X_1, \ldots, X_k\}$*. Let* N_i *be the public view of* X_i*, and let* L_i *be an event log of* X_i*. Then* $B_i = mpC(maxC(N_i))$ *is a best matching private view for* N_i *and* L_i*.*

Proof. Let $N_i' \in Pr(N_i)$ be a private view of N_i. We prove $\delta(L_i, N_i') \geq \delta(L_i, B_i)$, which implies $conf(L_i, env(N_i')) \leq conf(L_i, env(B_i))$ for conformance w.r.t. fitness according to Def. 15. By the choice of N_i' and Prop. 1, we conclude that $R(env(N_i')) \subseteq R(env(B_i))$. Let $\sigma \in L_i$ be a trace in the event log L_i. Then, we have $\Gamma_{\sigma, env(N_i')} \subseteq \Gamma_{\sigma, env(B_i)}$ by Def. 10 and $\delta(\lambda_{env(N_i')}(\sigma)) \geq \delta(\lambda_{env(B_i)}(\sigma))$ by Defs. 11 and 12. Thus, $\delta(L_i, N_i') \geq \delta(L_i, B_i)$ by Def. 15. □

Theorem 2 gives a theoretical solution for deciding private view conformance w.r.t. fitness. In addition, Thm. 2 gives a necessary condition for the question whether the implementation accords with the given public view N_i: If the best matching private view B_i does not conform to the event log L_i, then no private view of N_i conforms to L_i.

Corollary 3. *Let* N *be a public view,* L *be an event log of an implementation of* N*, and* B *be the best matching private view of* N*. If* B *does not conform to* L*, then no private view of* N *conforms to* L*.*

Of course, we are interested in calculating the best matching private view B_i for a given open net N_i. Here, we reuse existing theory on maximal controllers [14,8]. Interestingly, the environment (i.e., env^s or env^a) we consider when replaying the log file matters only for the construction of B_i. In the next section, we show that $B_i = mpC(maxC(N_i))$ can actually be calculated, yet for acyclic open nets only. The reason for this restriction is that for acyclic open nets, the correctness notions weak termination and deadlock freedom coincide. The theory for maximal controllers in case of weak termination exists [8], but has not been implemented so far.

5 Experimental Results

Based on a prototypical implementation, we show first experimental results on computing a canonical best matching private view according to Thm. 2. We assume weak termination as a correctness criterion, use the asynchronous environment env^a, and employ the standard distance function δ_S to find the best matching alignments.

For the running example, $\gamma_1 - \gamma_3$ are best matching alignments for L and $env(Public)$ with costs $\delta_S(\gamma_1) = 0$, $\delta_S(\gamma_2) = 2$, and $\delta_S(\gamma_3) = 1$, yielding alignment-based costs $\delta(L, env(Public)) = 30 \cdot 0 + 40 \cdot 2 + 50 \cdot 1 = 130$. Likewise, $\gamma_4 - \gamma_6$ are best matching alignments for L and $env(Private)$ with costs $\delta_S(\gamma_4) = \delta_S(\gamma_5) = 0$, and $\delta_S(\gamma_6) = 1$. Thus, $\delta(L, env(Private)) = 30 \cdot 0 + 40 \cdot 0 + 50 \cdot 1 = 50$.

We compute the canonical best matching private view B of *Public* in three steps: (1) compute the maximal controller $maxC(Public)$, (2) compute the most permissive controller $B = mpC(maxC(Public))$, and (3) calculate $\delta(L, env(B))$.

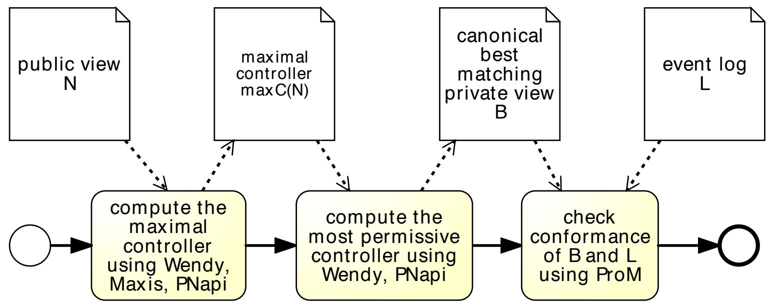


Fig. 6: Conformance checking using the best matching private view.

Figure 6 shows the three steps and the tools involved. Our toolchain consists of a Bash script for deriving a best matching private view using the tools Wendy [13], Maxis[1], the PNapi [11], and ProM.[2] We illustrate our approach in the following.

Step 1: Calculating $maxC(Public)$

The open net $maxC(Public)$ has 34 places and 45 transitions and was constructed following the approach presented in [14]: Using the tool Wendy, we constructed an annotated automaton that represents all controllers of *Public*. Subsequent, we derived the behavior of $maxC(Public)$ from this annotated automaton using the tool Maxis. Finally, we transformed the behavior into an open net using the PNapi. Figure 7 illustrates a part of $maxC(Public)$. As $maxC(Public)$ is a controller of *Public*, it has the same interface as *Public* with input and output interchanged. Initially, this service fires nondeterministically one of the five transitions $tabd, \ldots, td$. Depending on the state reached, it can perform a number of sending or receiving events. For example, after firing $tabd$, the open net can receive a or b or send d.

Step 2: Deriving B

In the second step, we calculated the most permissive controller of $maxC(Public)$, resulting in the open net $B = mpC(maxC(Public))$. We constructed the behavior of B using the tool Wendy and transformed it into open net B using the PNapi. The resulting open net has 12 places and 22 transitions and is partly depicted in Fig. 8. The open nets B and *Public* are interface-equivalent. Consider the place *empty*. A token on *empty* corresponds to a marking that is not reachable in the composition of B and any controller of *Public*. As no controller of *Public* initially sends a message c, transition $t8$ and $t9$ encode such “misbehavior” by producing a token on *empty*. When *empty* contains a token and hence the composition will not be weakly terminating, every possible sending

[1] `http://svn.gna.org/viewcvs/service-tech/trunk/maxis/`

[2] `http://www.promtools.org/`

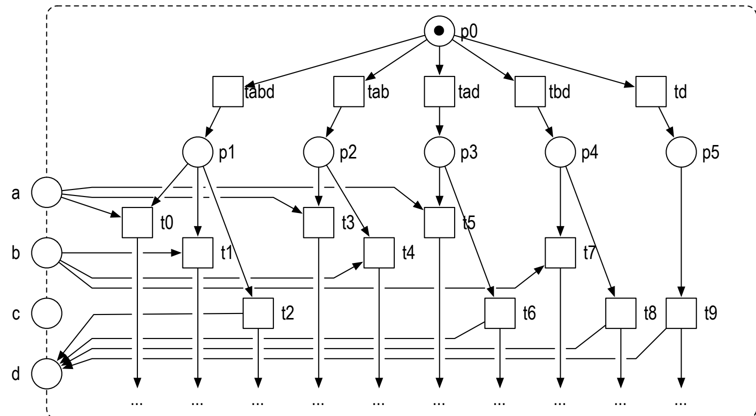


Fig. 7: The maximal controller $maxC(Public)$ of $Public$.

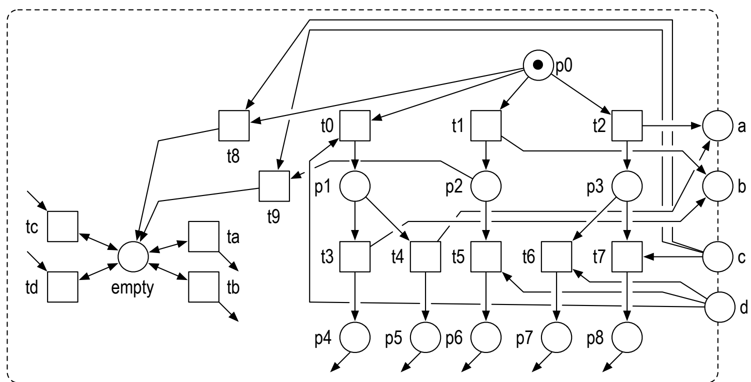


Fig. 8: The best matching private view $B = mpC(maxC(Public))$ of $Public$.

and receiving of messages is possible; thus, transitions ta, tb, tc, td are connected to the correspondingly labeled interface places (indicated by the respective arcs without source or target). What we can see is that the behavior of *Private* can be replayed on B. This shows that it is not wrong to implement a specification such that the resulting implementation has more controllers than the specification. However, the added behavior cannot be used by any controller of the specification. In our example, no controller of *Public* will initially send message c although there exist implementations such as open net *Private* that allow such behavior.

Step 3: Checking Conformance of L with B

According to Thm. 2, B is a best matching private view of *Public*. Therefore, in the last step, we calculate the alignment-based cost for the log L and the labeled

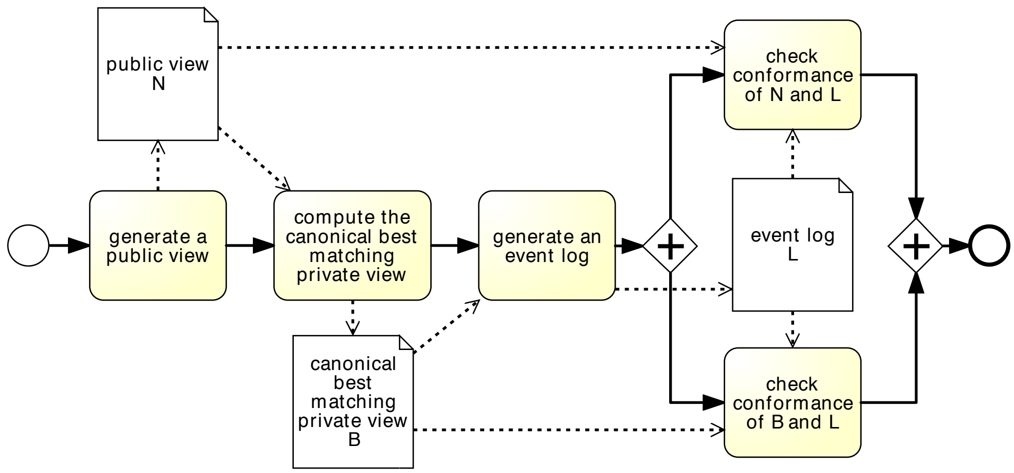


Fig. 9: Our evaluation process with synthetic nets.

net $env(B)$ using the latest PNAlignmentAnalysis plug-in from the TU/e SVN repository.[3] We use the A^*-algorithm for cost-based fitness with default options. Some best matching alignments for L and $env(B)$—as they are not unique—are

$$\gamma_7 = \begin{array}{|c|c|c|c|} \gg & a & d & \gg \\ \hline \tau & a & d & \tau \\ t_2 & t^a & t^d & t_6 \end{array} \quad \gamma_8 = \begin{array}{|c|c|c|c|} \gg & b & c & \gg \\ \hline \tau & b & c & \tau \\ t_1 & t^b & t^c & t_9 \end{array} \quad \gamma_9 = \begin{array}{|c|c|c|c|c|c|} c & \gg & d & \gg & \gg & a \\ \hline c & \tau & d & \tau & \tau & a \\ t^c & t_8 & t^d & td & ta & t^a \end{array}$$

with $\delta(\gamma_7) = \delta(\gamma_8) = \delta(\gamma_9) = 0$ yielding alignment-based costs $\delta(L, env(B)) = 30 \cdot 0 + 40 \cdot 0 + 50 \cdot 0 = 0$. We see that $\delta(L, env(B))$ is indeed lower than $\delta(L, env(Public)) = 130$ and even lower than $\delta(L, env(Private)) = 50$.

We also evaluated our approach with synthetic open nets. Figure 9 shows a BPMN model of our evaluation process. First, we generated a random public view N using a modified version of the Process Log Generator[4]. Afterward, we computed the canonical best matching private view B from N and generated a random event log L from B, which additionally contains random errors. Finally, we checked conformance of B to L and compared it with the conformance of N and L. We analyzed five random public views. This time, we used the synchronous environment env^s for computing the private view conformance with ProM. All experiments were conducted on a MacBook Pro, Intel Core i5 CPU with 2.4 GHz and 8 GB of RAM.

The results of our evaluation process in Table 1 show that the average cost $\delta_S(B, L)$ for each case (using the standard distance function) for conformance checking the log L with the best matching private view B (column 13) is significantly lower than the average cost $\delta_S(N, L)$ for conformance checking L with the

[3] `https://svn.win.tue.nl/repos/prom/`

[4] `http://www.processmining.it/sw/plg`

Table 1: Fully automatic private view conformance checking of synthetic nets.

public view N				best matching B				event log L		$\delta_S(N, L)$	time	$\delta_S(B, L)$	time
$\lvert P\rvert$	$\lvert I\rvert$	$\lvert O\rvert$	$\lvert T\rvert$	$\lvert P\rvert$	$\lvert I\rvert$	$\lvert O\rvert$	$\lvert T\rvert$	cases	events	$\delta_S/case$	$ms/case$	$\delta_S/case$	$ms/case$
14	4	2	6	35	4	2	132	100	605	6.21	3.47	0.20	0.34
16	5	3	8	41	5	3	190	100	541	7.53	3.31	0.20	0.88
30	6	3	18	106	6	3	681	100	540	8.26	6.21	0.19	1.41
38	6	4	32	32	6	4	168	100	507	4.89	7.10	0.05	0.17
88	6	5	74	806	6	5	6,060	100	528	7.24	33.93	0.03	45.60

public view N (column 11). This detail justifies Thm. 2. However, the lower cost come at a price of an exponentially larger size of B compared to N (columns 1 and 5), which is caused by the construction of B [14]. Accordingly, the larger net size resulted in a higher runtime of the A^*-algorithm (last row).

6 Related Work

Research on conformance checking of services follows two lines. One research line assumes a model of the implementation to be given (e.g., [20,6]) or that it is discovered from the event log (e.g., [15]). The former assumption is not always realistic. Furthermore, the result of conformance checking relies on the quality of the (discovered) model.

The second research line assumes recorded behavior of the implementation to be given. Here, techniques are adapted from process mining [18,2]. Our contribution follows this research line. Van der Aalst et al. [4] map a contract specified in BPEL onto workflow nets (which can be seen as the synchronous environment) and employ conformance checking techniques from process mining [18]. In contrast, we measure the deviation of an implementation from its specification and all possible private views.

Comuzzi et al. [7] investigate online conformance checking using a weaker refinement notion than accordance. Different conformance relations on a concurrency-enabled model have been studied by De León et al. [9]. As their considered conformance relations differ from accordance, their work is not applicable in our setting (because maximal controllers have not been studied yet).

Motahari-Nezhad et al. [16] investigate event correlation; that is, they try to find relationships between events that belong to the same process execution instance. In contrast to event correlation, we do not vary the service instances, but refine the public view to a private view.

7 Conclusion

Given a formal model of a public view of a service and recorded behavior of its running implementation, conformance checking requires to check the conformance of the recorded behavior with all (infinitely) private views of the specification. To overcome these infinitely many checks, we presented an approach to calculate a best matching private view for a given event log and a public view. Moreover, checking conformance of a best matching private view and a given event log from an implementation gives a necessary condition for accordance of this implementation with its public view. We proved the existence of a canonical best matching private view and showed that it can be automatically constructed—in the case of acyclic services and weak termination—using existing theory and tools on maximal controllers controllers. Although it is possible to construct maximal controllers for cyclic services and weak termination [8], this has not been implemented yet. For the actual conformance check, we used existing alignment-based techniques from the field of process mining.

A canonical best matching private view may become exponentially large in net size compared to its public view. Therefore, it is an open question whether the current cost-based conformance checking techniques can be used for private view conformance checking for industrial service models. In general, there exist many best matching private views for a public view w.r.t. the fitness dimension. Our approach computes a canonical best matching private view. There is a trade-off between the fitness dimension and the other quality dimensions (i.e., precision, generalization, simplicity) in conformance checking [2]; thus, it is an open question how to generalize our approach to these other dimensions.

References

1. Aalst, W.M.P.v.d.: Inheritance of Interorganizational Workflows: How to agree to disagree without loosing control? Information Technology and Management (2003)
2. 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)
3. Aalst, W.M.P.v.d., Dongen, B.F.v., Herbst, J., Maruster, L., Schimm, G., Weijters, A.: Workflow mining: A survey of issues and approaches. Data & Knowledge Engineering 47(2), 237267 (Nov 2003)
4. Aalst, W.M.P.v.d., Dumas, M., Ouyang, C., Rozinat, A., Verbeek, E.: Conformance checking of service behavior. ACM Transactions on Internet Technology 8(3) (2008)
5. Aalst, W.M.P.v.d., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: Multiparty contracts: Agreeing and implementing interorganizational processes. Comput. J. 53(1), 90–106 (2010)
6. Bravetti, M., Zavattaro, G.: Contract-based discovery and composition of web services. In: SFM 2009. LNCS, vol. 5569, pp. 261–295. Springer (2009)
7. Comuzzi, M., Vonk, J., Grefen, P.: Measures and mechanisms for process monitoring in evolving business networks. Data & Knowledge Engineering 71(1) (2012)
8. Hee, K.v., Mooij, A.J., Sidorova, N., Werf, J.M.v.d.: Soundness-preserving refinements of service compositions. In: WS-FM 2010. LNCS, vol. 6551, pp. 131–145. Springer (2011)
9. de León, H.P., Haar, S., Longuet, D.: Conformance relations for labeled event structures. In: TAP 2012. LNCS, vol. 7305, pp. 83–98. Springer (2012)
10. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: ICATPN 2007. LNCS, vol. 4546, pp. 321–341. Springer (2007)

11. Lohmann, N., Mennicke, S., Sura, C.: The Petri Net API A collection of Petri net-related functions. AWPN (2010)
12. Lohmann, N., Verbeek, E., Dijkman, R.: Petri net transformations for business processesa survey. Transactions on Petri Nets and Other Models of Concurrency II p. 4663 (2009)
13. Lohmann, N., Weinberg, D.: Wendy: A tool to synthesize partners for services. Fundam. Inform. 113(3-4), 295–311 (2011)
14. Mooij, A.J., Parnjai, J., Stahl, C., Voorhoeve, M.: Constructing replaceable services using operating guidelines and maximal controllers. In: WS-FM 2010. LNCS, vol. 6551, pp. 116–130. Springer (2011)
15. Motahari-Nezhad, H.R., Saint-Paul, R., Benatallah, B.: Deriving protocol models from imperfect service conversation logs. IEEE Transactions on Knowledge and Data Engineering 20(12), 1683–1698 (2008)
16. 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)
17. Papazoglou, M.: Web Services - Principles and Technology. Prentice Hall (2008)
18. Rozinat, A., Aalst, W.M.P.v.d.: Conformance checking of processes based on monitoring real behavior. Inf. Syst. 33(1), 64–95 (2008)
19. Stahl, C., Massuthe, P., Bretschneider, J.: Deciding substitutability of services with operating guidelines. In: ToPNoC II. pp. 172–191. LNCS 5460, Springer (2009)
20. Tran, H., Zdun, U., Dustdar, S.: Vbtrace: using view-based and model-driven development to support traceability in process-driven soas. Software & Systems Modeling 10(1), 5–29 (2009)
21. Vogler, W.: Modular Construction and Partial Order Semantics of Petri Nets, LNCS, vol. 625. Springer (1992)
22. Wolf, K.: Does my service have partners? In: ToPNoC II. pp. 152–171. LNCS 5460, Springer (2009)