Software and Systems Modeling (2023) 22:351–375
https://doi.org/10.1007/s10270-022-01008-x

REGULAR PAPER

Discovering architecture-aware and sound process models of multi-agent systems: a compositional approach

Roman Nesterov[1,2] · Luca Bernardinello[1] · Irina Lomazova[2] · Lucia Pomello[1]

Received: 3 May 2021 / Revised: 25 January 2022 / Accepted: 29 March 2022 / Published online: 3 May 2022

Abstract
A process model discovered from an event log of a multi-agent system often does not fully cover certain viewpoints of its architecture. We consider those concerned with the structure of a model explicitly reflecting agent behavior and interactions. The direct discovery from an event log of a multi-agent system may result in an unclear model structure and over-generalizations of agent behavior. We suggest applying a compositional approach that yields architecture-aware process models of multi-agent systems. An event log of a multi-agent system is filtered by the behavior of individual agents. Then, a multi-agent system model is a composition of agent models discovered from filtered logs. We use an intermediate model, called an interface pattern, specifying agent interactions and representing the architecture of a multi-agent system. We design a collection of specific interface patterns modeling typical agent interactions. An interface pattern provides an abstract specification of interactions and has a part corresponding to the behavior of each agent. We use structural transformations to map agent models discovered from filtered logs on the respective parts in an interface pattern. If such a mapping exists, we guarantee that a composition of agent models preserves their soundness. We conduct a series of experiments to evaluate the compositional approach. Experimental results confirm the improvement in the structure of process models discovered using the compositional approach compared to those discovered directly from event logs.

Keywords Multi-agent systems · Event logs · Process mining · Process discovery · Petri nets · Composition · Transformations · Interface patterns

1 Introduction

Modern information systems generate large amounts of event data, including, for example, transaction logs, message logs, and records of user activity. These data are commonly called *event logs*. They consist of ordered sequences (*traces*) of records on occurred events. Event logs are used in *process mining* to discover models of real processes [1]. The expected behavior of processes is usually specified manually at the early stages of the information system life cycle. Discovering the real behavior of processes from event logs is an important task since manually created models do not reflect changes made during the operation of an information system.

A wide range of algorithms supports the automated discovery of process models [2]. Process models can be represented in different notations. Process mining extensively uses various classes of Petri nets, heuristic nets, causal nets, and Business Process Model and Notation (BPMN). Our study focuses on modeling the *control-flow* of processes. We abstract from data used in the process execution. We choose Petri nets [3], a widely used formalism for modeling process behavior. Petri nets are also the basis for many other modeling notations, e.g., distinct classes of BPMN models can be transformed to Petri nets, and vice versa [4].

Communicated by Tony Clark.

This work is supported by MIUR, Italy and the Basic Research Program at HSE University, Russia.

✉ Roman Nesterov
rnesterov@hse.ru

Luca Bernardinello
luca.bernardinello@unimib.it

Irina Lomazova
ilomazova@hse.ru

Lucia Pomello
lucia.pomello@unimib.it

[1] University of Milano-Bicocca, Milan, Italy

[2] HSE University, Moscow, Russia

Table 1 Event log of a multi-agent system

Timestamp	Action	Agent
Trace 1		
30-12-2020:14.45	Register request	Pete
05-01-2021:09.34	Check ticket	John
07-01-2021:12.12	Examine causally	Pete
09-01-2021:10.15	Decide	John, Pete
12-01-2021:13.25	Pay compensation	Nick
Trace 2		
30-12-2020:16.45	Register request	Pete
04-01-2021:10.12	Examine thoroughly	Pete
06-01-2021:09.34	Check ticket	John
09-01-2021:09.19	Decide	John, Pete
10-01-2021:12.26	Reject request	Nick

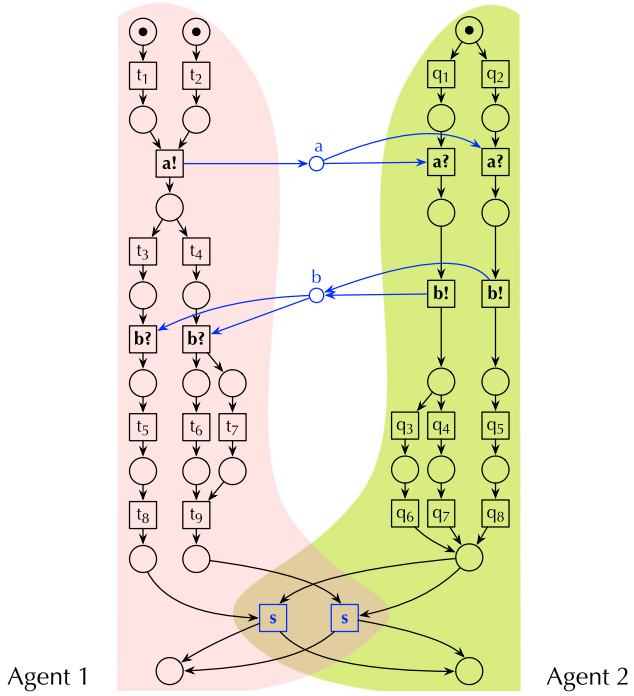


Fig. 1 Multi-agent system with two interacting agents

Four *conformance checking* dimensions, namely fitness, precision, generalization, and simplicity, determine the quality of process discovery algorithms [5]. *Fitness* estimates the extent to which a discovered process model can execute traces in an event log. A model *perfectly fits* an event log if it can execute all traces in an event log. *Precision* evaluates the ratio between the behavior allowed by a process model and not recorded in an event log. A model with perfect precision can only execute traces in an event log. The perfect precision limits the use of a process model since an event log represents only a finite "snapshot" of all possible process executions. Generalization and precision are two dual metrics. The fourth metric, *simplicity*, captures the structural complexity of a discovered model.

A record in an event log typically contains the name of an action and several additional attributes specifying the resources required for executing an action. For instance, in the event log shown in Table 1, an "Agent" attribute designates who has executed an action: *John*, *Pete*, or *Nick*. They execute actions independently, e.g., Pete registers a request, John checks a ticket, or together, e.g., John and Pete decide whether to pay the compensation. We say that an event log, where records contain information on agents, represents the behavior of a *multi-agent system*.

Event logs of multi-agent systems require the additional analysis of agent behavior and interactions. Otherwise, a process model discovered from an event log of a multi-agent system will not fully cover certain viewpoints of its architecture. The following motivating example briefly explains this problem. A model discovered from an event log of a multi-agent system may have relatively high precision, but its unclear structure does not reflect agent behavior and interactions.

Consider the Petri net shown in Fig. 1. Its structure is self-explanatory, i.e., there are two independent agents (colored areas in the figure) communicating via four distinguished nodes. Two places, a and b, are used as asynchronous channels to exchange messages. Two transitions marked with s correspond to actions executed simultaneously by Agents 1 and 2.

Simulating the behavior of the Petri net from Fig. 1, we generate an event log L of a multi-agent system with two agents. Applying, for instance, *Inductive miner* [38] to L, we discover the Petri net shown in Fig. 2. The Inductive miner allows us to guarantee the perfect fitness of a discovered model, i.e., it can execute all traces in an event log. This Petri net also demonstrates a high precision evaluation (0.73461). However, the structure of this model is not clear. The Inductive miner inserted many additional "silent" transitions (black boxes in Fig. 2) to connect blocks of actions executed by Agents 1 and 2. The Petri net shown in Fig. 2 does not correctly represent some key viewpoints of the architecture of a multi-agent system with two agents exchanging messages and executing synchronous actions.

The direct discovery of a model from an event log of a multi-agent system can also over-generalize individual agent behavior. For example, in the Petri net shown in Fig. 1, transition q_7 fires after transition q_4, while in the Petri net shown in Fig. 2, transition q_7 can fire after transitions q_3, q_4, or q_5. The concurrent execution of independent agents leads to a wide variety of possible traces recorded in an event log. However, a discovered model should not introduce inappropriate generalizations of agent behavior.

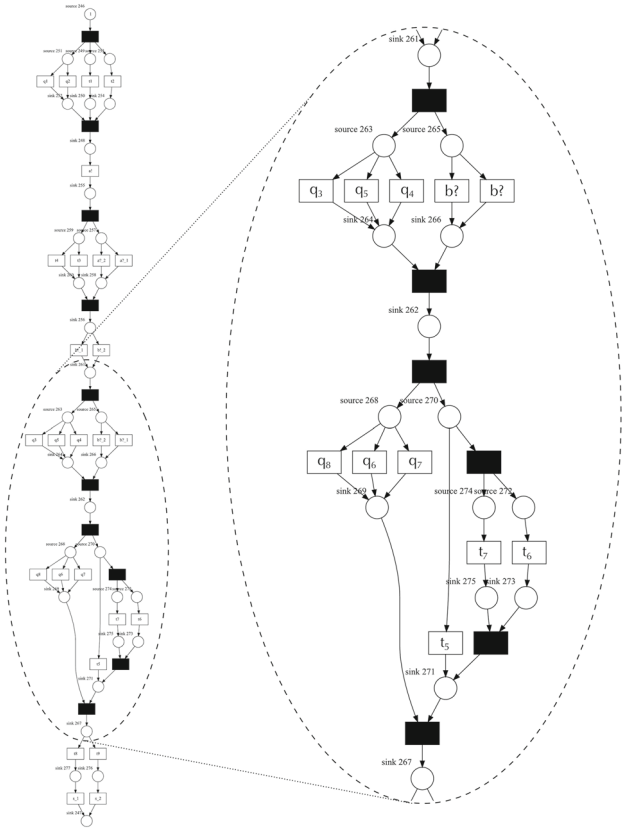


Fig. 2 Petri net discovered directly from the event log of the multi-agent system from Fig. 1

In this paper, given an event log of a multi-agent system, we aim to discover an *architecture-aware* Petri net, whose structure clearly covers the architecture viewpoints concerned with component interactions. In other words, a synthesized Petri net should explicitly show individual agent behavior and interactions similar to the Petri net shown in Fig. 1.

Various Petri net classes can be used to model the behavior of a multi-agent system. We will use *generalized workflow* (GWF) *nets* equipped with initial and final states. They differ from classical workflow nets [6] in allowing initial and final states to be sets of places rather than singletons. For instance, the Petri net shown in Fig. 1 is also a GWF-net with three initial places and two final places, while the behavior of Agent 2 is a classical workflow net.

We focus not only on the structural features of discovered GWF-nets but also on their behavioral properties. *Soundness* [7] is the fundamental correctness property of process behavior. Soundness is also called *proper termination*. A sound process can reach its final state from all intermediate states. The final state in a sound process cannot be contained in any other reachable state. Apart from that, a sound process does not have dead actions, which cannot be executed.

So we define the purpose of our study more precisely as follows. Given an event log of a multi-agent system and a specification of agent interactions, the task is to discover a *sound* and *architecture-aware* GWF-net, such that there are subnets showing agent behavior and nodes corresponding to agent interactions.

A specification of agent interactions is called an *interface*. It represents the key interaction-oriented viewpoints of the architecture of a multi-agent system. We suppose that an interface is provided by experts in advance, e.g., system architects may construct candidate interfaces. The adequacy of these candidates can be determined by checking their conformance to an event log. Our paper does not consider discovering an interface model directly from an event log of a multi-agent system. Therefore, we *assume* that:

1. All records in an event log have the corresponding "Agent" attribute.
2. There is a distinguished set of actions through which agents communicate via message exchange and synchronizations. For instance, in the event log shown in Table 1, the "decide" action is executed simultaneously by John and Pete.

We propose a *compositional* approach that allows us to discover sound and architecture-aware process models of multi-agent systems. Even a simple composition of sound models might not be sound, e.g., it can have a deadlock. That is why we do not consider arbitrary interfaces. The main idea of our solution is to choose specific *interface patterns*, which preserve agent soundness, and formulate the conditions for a correct application of these patterns. Similar *service interaction patterns* are used in Business Process Management for a correct organization of communication in large-scale information systems [8]. Service interaction patterns represent *typical* communication scenarios. We use them to design our collection of interface patterns. An interface pattern is a GWF-net that:

– Provides a highly *abstract* view of agent interactions without exposing internal agent behavior;
– Has a part representing the behavior of each agent.

The *central hypothesis* of our study is that the compositional approach improves the quality of discovered process models compared to the quality of process models discovered directly from event logs of multi-agent systems.

An algorithm of the compositional process discovery includes three steps, as shown in Fig. 3. These steps are discussed below.

Filtration An event log of a multi-agent system is filtered by actions executed by different agents. Correspondingly, we construct a set of *sub-logs*. For instance, filtering the records in the event log given in Table 1 by the "Pete" value of the "Agent" attribute, we obtain the sub-log presented in Table 2.

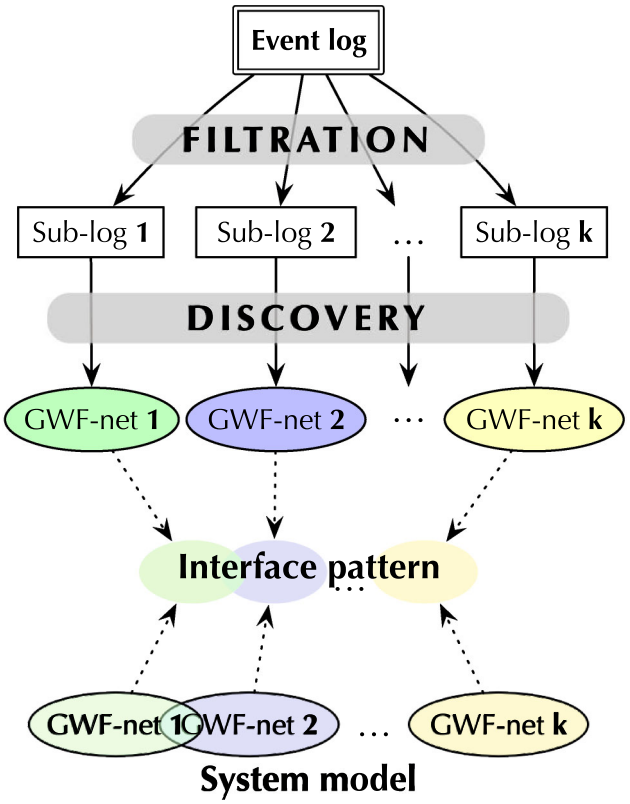


Fig. 3 Compositional process discovery

Table 2 Pete's sub-log of the event log from Table 1

Timestamp	Action	Agent
Trace 1		
30-12-2020:14.45	Register request	Pete
07-01-2021:12.12	Examine causally	Pete
09-01-2021:10.15	Decide	Pete
Trace 2		
30-12-2020:16.45	Register request	Pete
04-01-2021:10.12	Examine thoroughly	Pete
09-01-2021:09.19	Decide	Pete

Discovery We discover agent GWF-nets from corresponding sub-logs constructed at the filtration step. Discovered GWF-nets should be sound. The *Inductive miner*, mentioned above, discovers sound models.

Composition If there is a mapping of an agent GWF-net to the corresponding part in an interface pattern (shown by dashed arcs in Fig. 3), we can replace this abstract part with the agent GWF-net. As a result, we obtain a sound process model of a multi-agent system, provided that we manage to find a mapping for every agent GWF-net.

The following three aspects determine the correctness of the compositional process discovery algorithm:

1. Theoretical backgrounds of an event log filtration and a GWF-net composition.
2. Interface patterns describing soundness-preserving interactions among agents in a multi-agent system.
3. A technique to map agent models discovered from filtered sub-logs on the corresponding parts in an interface pattern.

We discuss these aspects step by step in the paper.

Thus, our study makes the following *contributions*:

1. It proposes an algorithm for discovering sound and architecture-aware models of multi-agent systems.
2. It proposes a collection of sound interface patterns representing typical agent interactions.
3. It proves the correctness of the proposed algorithm.
4. It conducts a thorough experimental evaluation of the proposed algorithm according to the main hypothesis of our study.

The remainder of our paper is organized as follows. In the next section, we formally define event logs of multi-agent systems and GWF-nets. Section 3 presents a collection of sound interface patterns that are used to model agent interactions in the compositional process discovery. In Sect. 4, we design an algorithm of the compositional process discovery and prove its correctness. In Sect. 5, we present and discuss the outcomes from experiments conducted to evaluate the compositional process discovery. Section 6 discusses related research. Finally, Sect. 7 concludes the paper and suggests future work directions.

2 Theoretical backgrounds of compositional process discovery

The first correctness aspect of the compositional approach to process discovery is its formal basis. This section collects the definitions of general notions, event logs, and (a composition of) generalized workflow nets. We refer to them when modeling interface patterns in Sect. 3 and proving the correctness of the compositional process discovery algorithm in Sect. 4.

S^+ denotes the set of all finite non-empty sequences over a finite non-empty set S, and $S^* = S^+ \cup \{\varepsilon\}$ where ε is the empty sequence. Let $\sigma \in S^*$ and S' be a subset of S. Then $\sigma|_{S'}$ denotes the *projection* of σ on S'. In other words, $\sigma|_{S'}$ is the subsequence of σ obtained by removing elements not belonging to S'. For example, let $S = \{a, b, c, d\}$, $\sigma = abadabcdcb \in S^*$, and $S' = \{b, c\}$. Projecting σ on S' gives $\sigma|_{S'} = bbccb$.

$\mathbb{N}$ denotes the set of non-negative integers. A function $m\colon S \to \mathbb{N}$ defines a *multiset* m over a non-empty set S. We write $s \in m$ iff $m(s) > 0$. The set of all finite multisets over S

is denoted by $\mathcal{B}(S)$. Let $m_1, m_2 \in \mathcal{B}(S)$. Then $m_1 \subseteq m_2$ iff $m_1(s) \leq m_2(s)$; $m' = m_1 \cup m_2$ iff $m'(s) = m_1(s) + m_2(s)$; $m'' = m_1 \setminus m_2$ iff $m''(s) = \max(m_1(s) - m_2(s), 0)$ for all $s \in S$.

2.1 Event logs and log projections

An event log is the main data source used to discover process models. It is a multiset of *traces* — ordered sequences of actions.

Definition 1 (*Log*) Let Λ denote the set of all actions. A trace is a finite non-empty sequence σ over Λ, i.e., $\sigma \in \Lambda^+$. An event log L over Λ is a multiset of traces, i.e., $L \in \mathcal{B}(\Lambda^+)$.

Given an event log L over Λ and $\Lambda' \subseteq \Lambda$, we can project L on Λ' by projecting all traces in L on Λ'. We need to take into account only non-empty projections of traces in L as well as the fact that the projections of different traces in L may coincide.

Definition 2 (*Log projection*) Let $L \in \mathcal{B}(\Lambda^+)$ be an event log and $\Lambda' \subseteq \Lambda$. The projection of L on Λ' is an event log, denoted $L_{\Lambda'} \in \mathcal{B}((\Lambda')^+)$, containing non-empty projections of traces in L on Λ' such that:

$$\forall \sigma \in L\colon \sigma|_{\Lambda'} \in L_{\Lambda'} \Leftrightarrow \sigma|_{\Lambda'} \neq \varepsilon \text{ and}$$
$$\forall \sigma \in L\colon \sigma|_{\Lambda'} = \sigma' \in L_{\Lambda'}\colon L_{\Lambda'}(\sigma') = \sum L(\sigma).$$

According to the main assumptions of the compositional process discovery, actions in an event log of a multi-agent system are assigned agents executing them. Then, Λ can be decomposed into k (possibly overlapping) subsets of actions correspondingly, i.e., $\Lambda = \Lambda_1 \cup ... \cup \Lambda_k$. Moreover, there is a distinguished subset $In \subseteq \Lambda$ of actions through which agents interact. *In* is also called a set of *interacting* actions.

Then, the discovery of an individual agent model from an event log of a multi-agent system L involves projecting L, by Definition 2, on a corresponding subset of actions Λ_i, i.e., constructing L_{Λ_i} with $i = 1, 2, ..., k$. Log projections are also called *sub-logs*. For example, the sub-log shown in Table 2 is obtained from the log shown in Table 1 by removing actions not executed by Pete from its traces.

2.2 Generalized workflow nets

Workflow (WF) nets [6] are basic models used in process discovery. A WF-net is a Petri net with a distinguished initial and final place. The execution of a trace in an event log corresponds to the execution of a WF-net from its initial to its final place. For a more convenient representation of multi-agent systems, we generalize WF-nets, allowing sets of initial and final places rather than singletons. Here, we define *generalized workflow nets* and their behavior.

Definition 3 (*Net*) A *net* is a triple $N = (P, T, F)$ where P and T are two disjoint sets of places and transitions, and $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation. For any node $x \in P \cup T$:

1. ${}^\bullet x = \{y \in X \mid (y, x) \in F\}$ is the *preset* of x.
2. $x^\bullet = \{y \in X \mid (x, y) \in F\}$ is the *postset* of x.
3. ${}^\bullet x^\bullet = {}^\bullet x \cup x^\bullet$ is the *neighborhood* of $x \in X$.

A net is called *P-simple* if $\forall p_1, p_2 \in P\colon {}^\bullet p_1 = {}^\bullet p_2$ and $p_1{}^\bullet = p_2{}^\bullet$ implies $p_1 = p_2$. In our work, we consider nets without self-loops, i.e., $\forall x \in P \cup T\colon {}^\bullet x \cap x^\bullet = \varnothing$ and isolated transitions, i.e., $\forall t \in T\colon |{}^\bullet t| \geq 1$ and $|t^\bullet| \geq 1$.

The $\bullet$-notation is also extended to subsets of nodes. Let $N = (P, T, F)$ be a net, and $Y \subseteq P \cup T$. Then ${}^\bullet Y = \bigcup_{y \in Y} {}^\bullet y$, $Y^\bullet = \bigcup_{y \in Y} y^\bullet$ and ${}^\bullet Y^\bullet = {}^\bullet Y \cup Y^\bullet$. $N(Y)$ denotes the subnet of N *generated* by Y, i.e., $N(Y) = (P \cap Y, T \cap Y, F \cap (Y \times Y))$.

A *marking* (state) m in a net $N = (P, T, F)$ is a multiset over P, i.e., $m\colon P \to \mathbb{N}$. Marking m is *safe* iff $\forall p \in P\colon m(p) \leq 1$, i.e., a safe marking is a set of places. Marking m of place $p \in P$ is depicted by putting $m(p)$ black dots inside p.

Definition 4 (*Net system*) A *net system* a quadruple $N = (P, T, F, m_0)$ where (P, T, F) is a net, and $m_0\colon P \to \mathbb{N}$ is the *initial* marking.

A marking m in a net $N = (P, T, F)$ enables transition $t \in T$, denoted $m[t\rangle$, iff ${}^\bullet t \subseteq m$. Enabled transitions may *fire*. Firing t at m evolves N to a new marking $m' = (m \setminus {}^\bullet t) \cup t^\bullet$, denoted $m[t\rangle m'$.

A sequence $w \in T^*$ is a *firing sequence* in a net system $N = (P, T, F, m_0)$ if $w = t_1 t_2 ... t_n$ and $m_0[t_1\rangle m_1 [t_2\rangle ... m_{n-1}[t_n\rangle m_n$. Then we write $m[w\rangle m_n$. The set of all firing sequences in N is denoted by $FS(N)$.

A marking m in $N = (P, T, F, m_0)$ is *reachable* if $\exists w \in FS(N)\colon m_0[w\rangle m$. Any marking can be reached from itself by the empty sequence, i.e., $m[\varepsilon\rangle m$. The set of all markings reachable from m is denoted by $[m\rangle$. N is safe iff all reachable markings in N are safe.

A *state machine* is a connected net (P, T, F), where $\forall t \in T\colon |{}^\bullet t| = |t^\bullet| = 1$. A subnet of $N = (P, T, F, m_0)$ generated by $Y \subseteq P$ and ${}^\bullet Y^\bullet$, i.e., $N(Y \cup {}^\bullet Y^\bullet)$, is a *sequential component* of N if it is a state machine and has a single token in the initial marking. N is covered by sequential components if every place belongs to at least one sequential component. In this case, N is *state machine decomposable* (SMD).

State machine decomposability is a basic feature bridging structural and behavioral properties of nets, also considered in [6] as an important feature of workflow nets. It is easy to see that SMD net systems are safe since their initial markings are safe. We further work with SMD net systems,

unless otherwise stated explicitly. Thus, we omit SMD in their descriptions.

In a GWF-net, we impose additional restrictions on its initial marking and distinguish its final marking.

Definition 5 (*GWF-net*) A *generalized workflow net* is a net system $N = (P, T, F, m_0)$ equipped with the *final* marking $m_f \subseteq P$ such that:

1. ${}^\bullet m_0 = \varnothing$.
2. $m_f{}^\bullet = \varnothing$.
3. $\forall x \in P \cup T\ \exists s \in m_0\ \exists f \in m_f\colon (s, x), (x, f) \in F^*$ where F^* is the reflexive transitive closure of F.

According to the third requirement in Definition 5, any node in a GWF-net lies on a path from a place in its initial marking to a place in its final marking.

Soundness is the main correctness property of the process behavior formally specified by a GWF-net. Different kinds of soundness were studied in [7]. Here, we use the *classical soundness* related to the reachability of the final marking in a GWF-net.

Definition 6 (*Soundness*) A GWF-net $N = (P, T, F, m_0, m_f)$ is *sound* if and only if:

1. $\forall m \in [m_0\rangle\colon m_f \in [m\rangle$.
2. $\forall m \in [m_0\rangle\colon m_f \subseteq m \Rightarrow m = m_f$.
3. $\forall t \in T\ \exists m \in [m_0\rangle\colon m[t\rangle$.

Soundness is directly connected with the *proper termination* of a corresponding process. Every execution in a properly terminating process must finish in its final state, such that this final state is not contained in any other reachable state. Also, there must not be non-executable actions.

Event logs record the *observable* behavior of an information system. The observable behavior of a GWF-net $N = (P, T, F, m_0, m_f)$ is derived via transition *labels*. A total function $\ell\colon T \to \Lambda \cup \{\tau\}$ is a *transition labeling* function where $\tau \notin \Lambda$ is the label of the *invisible* action. Given a firing sequence $tw \in FS(N)$, an observable execution $\ell(tw)$ is defined by (1) $\ell(tw) = \ell(t)\ell(w)$ if $\ell(t) \neq \tau$ and (2) $\ell(tw) = \ell(w)$ if $\ell(t) = \tau$.

2.3 Composition of GWF-nets

Earlier in [9], we defined a composition of GWF-nets interacting via message exchange and synchronizations. We use this composition to design interface patterns representing the architecture of a multi-agent system. Here, we briefly show how to compose interacting GWF-nets and recall the main properties.

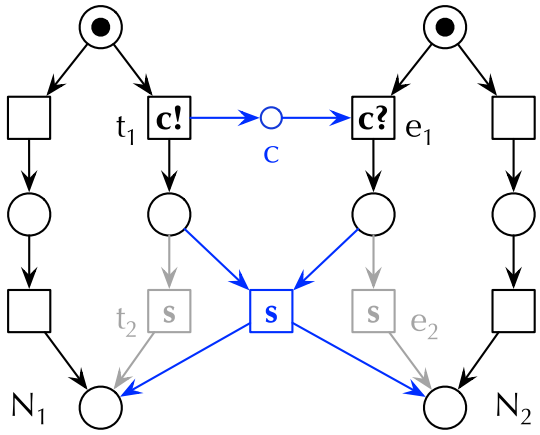


Fig. 4 Composition of two GWF-nets N_1 and N_2

Two kinds of transition labels model *asynchronous* and *synchronous* interactions among agents, whose behavior is formally represented via GWF-nets:

1. Asynchronous labels that carry *channel* names and operations. For instance, label "a!" corresponds to sending a message to channel a, and its *complement*, label "a?", corresponds to receiving a message from channel a.
2. Synchronous labels that represent a name of a corresponding action to be executed simultaneously by several agents.

These transition labels specify interactions among agents and define how GWF-nets representing their behavior are to be composed. These labels correspond to a set of interacting actions, denoted *In*, from an event log of a multi-agent system. Thus, this composition involves the two following operations:

1. *Insertion of channel places* connecting transitions with complement asynchronous labels.
2. *Pair-wise fusion* of transitions with the same synchronous labels.

Figure 4 illustrates the insertion of a place between transition t_1 in N_1 with label "c!" and transition e_1 in N_2 with label "c?". The added place is assigned the corresponding asynchronous label c. Figure 4 also shows the fusion of transition t_2 in N_1 and transition e_2 in N_2 since they have the same label s.

In [9], we studied the mathematical framework of this GWF-net composition. The composition of two GWF-nets with transition labels, denoted by $N_1 \circledast N_2$, is also a GWF-net. In addition, the composition of GWF-nets is commutative and associative. Thus, it is a convenient tool to model multi-agent systems with three or more interacting agents.

3 Framework of proposed interface patterns

The second correctness aspect of the compositional approach to process discovery is the proper specification of agent interactions. In this section, we study interfaces preserving the soundness of interacting agents.

It is easy to arrange agent interactions leading to deadlocks. For instance, consider, again, the GWF-net shown in Fig. 4. The two agent GWF-nets, N_1 and N_2, are sound in isolation. However, their composition is no longer sound since N_2 may decide not to receive a message from channel c. As a result, N_1 will not be able to synchronize further with N_2.

Therefore, we do not work with arbitrary interfaces in the compositional process discovery. We design specific *interface patterns* — ready-to-use interface models, describing *typical* and basic agent interactions. Moreover, interface patterns preserve the soundness of agent GWF-nets discovered from filtered sub-logs.

Firstly, we consider the classification and informal representation of patterns. Secondly, we formally specify interface patterns using the GWF-net composition.

3.1 Classification

Patterns are traditionally used in software engineering, e.g., *software design patterns* [10]. W. van der Aalst et al. [11] first introduced *workflow patterns* in Business Process Management to consolidate recurrent scenarios in the control-flow of business processes. Later, A. Barros et al. [8] generalized these patterns to model *typical* service interactions in large-scale systems. We take their classification of service interaction patterns and recall it below.

Interface patterns are distinguished by the number of interacting parties:

- *Bilateral* patterns specifying interactions between a pair of agents;
- *Multilateral* patterns specifying interactions among three or more agents.

Interface patterns are also classified according to the way agents interact:

- *Single transmission* patterns;
- *Multiple transmission* patterns.

The number of transmissions specifies how many times interacting agents can exchange messages. Multiple transmission patterns imply repeated message exchange that should have a possibility to be terminated to preserve the soundness of agent behavior.

Interface patterns should include three main parts, indicated by the scheme shown in Fig. 5. They are:

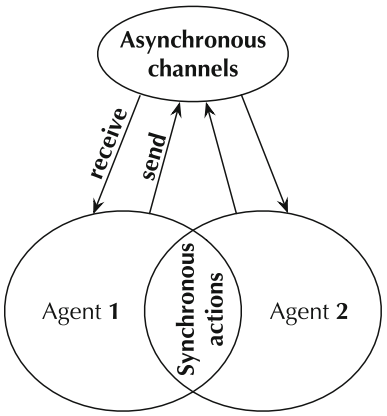


Fig. 5 Components of an interface pattern

1. The number of interacting agents.
2. How agents exchange messages via channels.
3. How and when agents execute synchronous actions.

Following this scheme will allow us to effortlessly translate the informal description of an interface pattern into a composition of GWF-nets.

While describing interface patterns, we also follow the general principles in the component-based design of information systems [12]:

1. An interface should provide enough information to establish correct communication among agents.
2. An interface should not expose the internal behavior of agents not required for their communications.

3.2 Informal representation

We design a set of interface patterns describing asynchronous and mixed asynchronous-synchronous interactions among agents, such that the soundness of their behavior is preserved. Using specific synchronous patterns in isolation is not of great value in modeling systems with complex agent interactions. That is why we will further consider different combinations of asynchronous message exchange and synchronizations.

Tables 3 and 4 give an informal representation of twelve interface patterns we use to represent the interaction-oriented architecture viewpoints of a multi-agent system in the compositional process discovery. A pattern contains dummy agent names and crucial aspects of agent interactions, according to the scheme from Fig. 5. We use identifiers to refer to these interface patterns further in the text.

Table 3 considers interface patterns developed using service interaction patterns presented in [8]. Single transmission patterns, IP-1, IP-2, and IP-3, describe rather primitive agent interactions since a sending agent is not supposed to receive

Table 3 Description of asynchronous interface patterns

Pattern	ID	Description
Send (Receive)	IP-1	An agent X sends (receives) a message to (from) an agent Y
Concurrent send (Receive)	IP-2	An agent X concurrently sends (receives) several messages (>1) to (from) an agent Y
Alternative send (Receive)	IP-3	An agent X sends (receives) exactly one out of two (or more) alternative message sets to (from) an agent Y
Exchange	IP-4	An agent X sends a message to an agent Y. Subsequently, Y sends a response to X
Concurrent exchange	IP-5	An agent X concurrently sends several messages (>1) to an agent Y. Then Y sends a response to each message received from X
Alternative exchange	IP-6	An agent X sends exactly one out of two (or more) alternative message sets to an agent Y. Subsequently, Y sends a corresponding response to a message received from X
Multiple exchange	IP-7	An iterative implementation of IP-4, such that the message exchange continues till an Agent X does not need responses from an Agent Y
Racing incoming messages	IP-8	An agent X receives one among a set of messages incoming from two or more other agents.

Table 4 Description of mixed interface patterns

Pattern	ID	Description
Sync before exchange	IP-9	Before exchanging messages, agents X and Y execute a synchronous action
Sync after exchange	IP-10	Agents X and Y execute a synchronous action after they exchange messages
Sync and exchange	IP-11	Concurrently with message exchange, agents X and Y execute a synchronous action
Sync or exchange	IP-12	Agents X and Y either execute a synchronous action or exchange messages but not both

an acknowledgment from the other agent. Various ways of asynchronous message exchange are given in patterns IP-4, IP-5, and IP-6. Interface pattern IP-7 describes multiple transmission interactions when one agent can decide to stop the exchange by sending a corresponding message to the other agent.

Multilateral interactions among three or more agents are described in IP-8. According to the specification of this pattern, one of the agents expects to receive one of several messages incoming from the other agents. Sending agents should be properly notified whether their messages are received.

Table 4 describes mixed interface patterns. They combine asynchronous and synchronous agent interactions. Patterns IP-9 and IP-10 extend pattern IP-4 such that agents synchronize either before or after messages are exchanged. Pattern IP-11 extends pattern IP-5 such that agents synchronize and exchange messages concurrently. Pattern IP-12 allows agents to either execute a synchronous activity or exchange messages. This corresponds to an extension of pattern IP-6.

The next step is to translate these informal descriptions of interface patterns into GWF-nets.

3.3 Formal specification

Figure 6 provides eight GWF-nets constructed according to the informal description of eight asynchronous interface patterns. Figure 7 provides four GWF-nets constructed according to the description of four interface patterns, combining both asynchronous and synchronous interactions. We discuss some important features of these models in more detail below.

Every interface pattern is a composition of GWF-nets representing abstractions of agent behavior. For example, in the GWF-net of pattern IP-1 shown in Fig. 6(a), abstract representations of agent behavior, A_1 and A_2, contain a single labeled transition used to send/receive a message. However, abstractions of agent GWF-nets can also contain transitions not labeled by interacting actions. They are required to model the specific control-flows of agents. For instance, the GWF-nets of patterns IP-2, IP-5, IP-8, and IP-11 (see Fig. 6(b), (e), (h), and Fig. 7(c) correspondingly) contain transitions used to model the splits and joins of parallel branches in agent behavior.

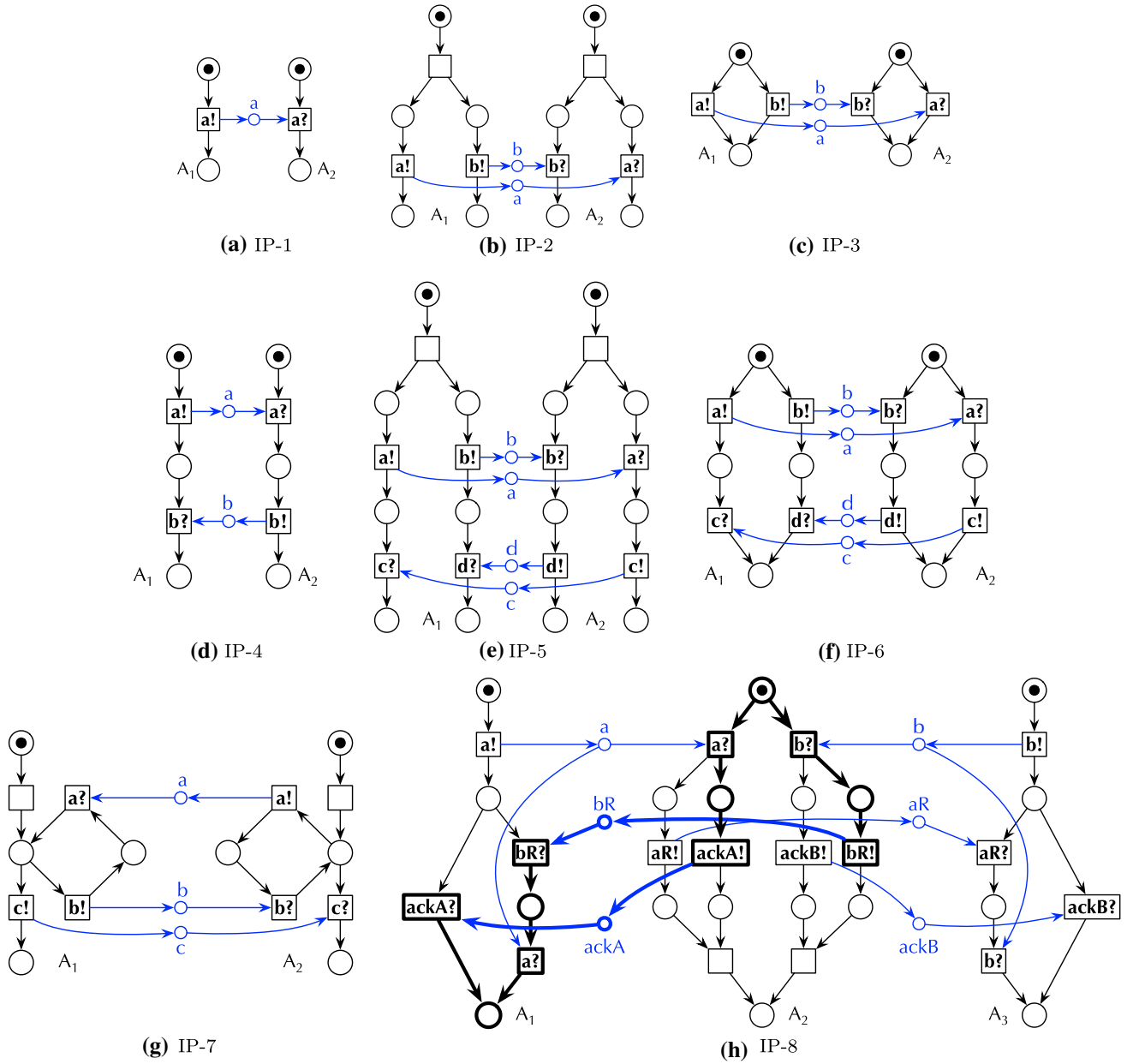
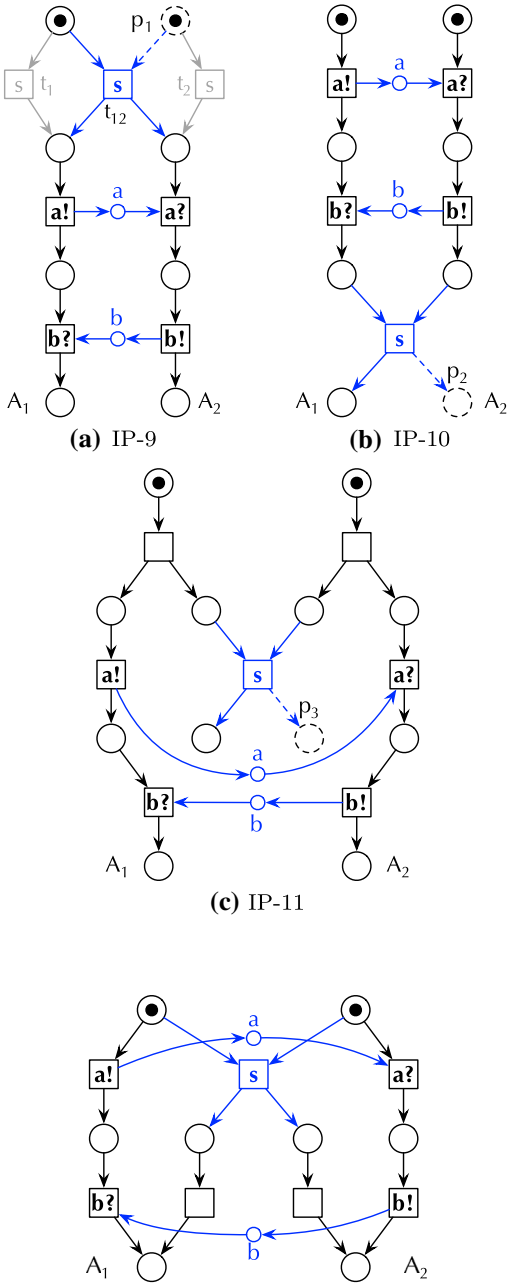


Fig. 6 Asynchronous interface patterns: GWF-nets

In the GWF-nets of single transmission interface patterns, IP-1, IP-2, and IP-3, channels are added only "in a single direction" to send/receive messages. According to the specification of these patterns, acknowledgments are not expected.

The remainder of bilateral asynchronous interface patterns contains different message exchange variations involving one channel to send a message and the other channel to send an acknowledgment. For example, in the GWF-net of pattern IP-5 shown in Fig. 6(e), there are two concurrent message exchanges between A_1 and A_2. A_1 sends a message to A_2 via channel a, and A_2 sends an acknowledgment to A_1 via channel c. Channels b and d are used similarly.

Consider the GWF-net of the multilateral pattern IP-8 with three agents, A_1, A_2, and A_3, as shown in Fig. 6(h). It has the most sophisticated structure among all asynchronous interface patterns. However, it has a clear interpretation. According to the specification of pattern IP-8 from Table 3, A_2 expects to receive one of two messages incoming from A_1 and A_3 through channels a and b correspondingly. Depending on which message is received (the one sent by A_1 or by A_3), A_2 executes the following actions:



(a) IP-9 **(b)** IP-10

(c) IP-11

(d) IP-12

Fig. 7 Mixed asynchronous-synchronous interface patterns: GWF-nets

1. It notifies A_1 (A_2) by sending an acknowledgment through channel $ackA$ ($ackB$) that the corresponding message is received;
2. It notifies the agent whose message is not received by sending a message to channel aR or channel bR.

A message left in channel a (b) is removed by the sending agent A_1 (A_3) to preserve the soundness of agents. In addition, a bold subnet in Fig. 6(h) corresponds to one of several sequential components covering the GWF-net of pattern IP-8. A similar analysis on sequential component decomposition can be done for all GWF-nets provided in Figs. 6 and 7.

As discussed earlier, we extend asynchronous interface patterns by introducing synchronizations into the structure of corresponding GWF-nets. In the GWF-nets of patterns IP-9 and IP-10 shown in Fig. 7(a), (b), synchronous action s is added before and after the message exchange via channels a and b to extend pattern IP-4. Also, in the GWF-nets of patterns IP-11 and IP-12 shown in Fig. 7(c), (d), synchronous action s replaces one of two branches of the message exchange, initially present in asynchronous interface patterns IP-5 and IP-6.

It is also important to note that pair-wise fusion of transitions can lead to *redundant* places that cannot be distinguished by their neighborhoods. Redundant places are identified in GWF-nets of patterns IP-9, IP-10, and IP-11. They are highlighted by dashed circles: place p_1 in Fig. 7(a), place p_2 in Fig. 7(b) and place p_3 in Fig. 7(c). These places can be safely removed to make a corresponding GWF-net P-simple.

The interface patterns discussed in this section describe interactions among agents in a multi-agent system such that the soundness of agent behavior is not violated. Therefore, Proposition 1 holds.

Proposition 1 *Interface pattern GWF-nets IP-1, IP-2, ..., IP-12 are sound.*

The proof of Proposition 1 is the straightforward verification of the requirements imposed by Definition 6. Note that the collection of interface patterns presented above is incomplete. One may further extend it, provided that an extended version of Proposition 1 holds for the new patterns as well.

4 Proposed algorithm for compositional process discovery

This section presents the main algorithm for the compositional discovery of process models from event logs of multi-agent systems where agent interactions are specified using interface patterns proposed in Sect. 3. We also discuss the third correctness aspect of the compositional process discovery. We propose a technique to map the models of individual agents discovered from filtered sub-logs on the corresponding subnets in an interface pattern.

We prove the correctness of the proposed algorithm from two perspectives:

1. A model of a multi-agent system discovered by this algorithm can execute all traces in an event log.

2. A model of a multi-agent system discovered by this algorithm is a sound GWF-net.

4.1 Algorithm

The compositional process discovery algorithm (see Algorithm 1) reflects the main steps of the general scheme of the approach shown in Fig. 3:

1. DISCOVER(L_{Λ_i}) corresponds to the application of the process discovery algorithm to agent sub-logs. It is important to obtain sound GWF-nets at this step. For instance, the Inductive miner [13] guarantees the soundness of discovered models.
2. ISREFINEMENT(R_i, A_i) checks if the agent GWF-net, R_i, is a proper refinement of the corresponding part, A_i, in the interface pattern.
3. REPLACE(S, A_i, R_i) substitutes the corresponding part, A_i, in the interface pattern with the agent GWF-net, R_i, discovered from L_{Λ_i}.

Algorithm 1: Compositional discovery

Input: L – an event log over $\Lambda = \Lambda_1 \cup ... \cup \Lambda_k \cup In$,
$IP = A_1 \circledast A_2 \circledast ... \circledast A_k$ – an interface pattern
Output: S – a multi-agent system GWF-net

```
S ← IP
foreach Λ_i ⊆ Λ do
  | R_i ← DISCOVER(L_{Λ_i})
end
ℜ ← {R_1, R_2, ..., R_k};
foreach R_i ∈ ℜ do
  | if ISREFINEMENT(R_i, A_i) then
  |   | REPLACE(S, A_i, R_i)
  | end
end
```

If all agent GWF-nets discovered from sub-logs are proper refinements of the corresponding parts in the interface pattern IP, we will obtain a complete multi-agent system model S.

However, it is also possible that only some GWF-nets discovered from filtered sub-logs are proper refinements of an interface pattern. For instance, given $IP = A_1 \circledast A_2$, we may obtain that R_1 is not a refinement of A_1, while R_2 is a refinement of A_2. Then a pattern will only be partially refined, and a system model $S = A_1 \circledast R_2$ will be an *approximation* of the model sought for. In addition, if none of the GWF-nets discovered from sub-logs are proper refinements of the interface pattern, then this algorithm will not change an interface pattern. In these cases, we may recommend modifying IP or developing a new interface pattern.

The correctness of Algorithm 1 is justified by the fact that a multi-agent system GWF-net S is sound and perfectly fits an event log L, provided that all agent GWF-nets can be mapped on an interface pattern. Further, we formalize and prove these properties.

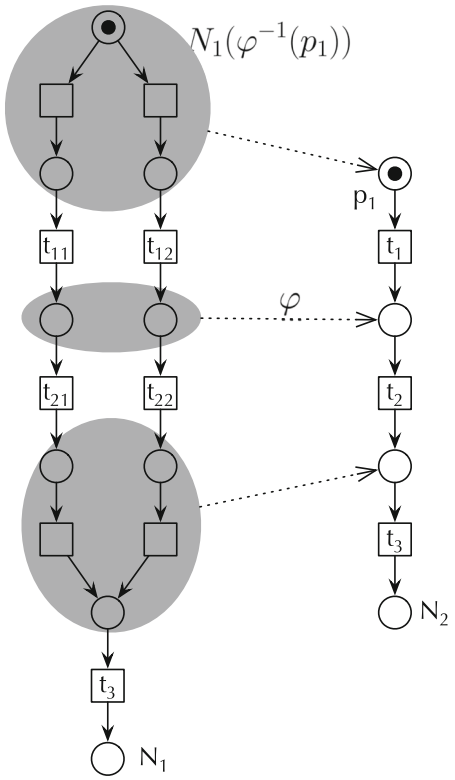


Fig. 8 An α-morphism $\varphi\colon N_1 \to N_2$

4.2 Refinement of GWF-nets

Here, we discuss the formal framework for the correct refinement check, ISREFINEMENT, that constitutes the basis of Algorithm 1, when a process model of a multi-agent system is actually constructed.

Abstraction/refinement relations between two GWF-nets are formalized with the help of α-morphisms introduced in [14]. Using the example shown in Fig. 8, we discuss the basic intuition behind α-morphisms.

An α-morphism, denoted by $\varphi\colon N_1 \to N_2$, is a total surjective map, where N_1 is called a *refinement* of N_2, and N_2 is called an *abstraction* of N_1. Places in N_2 can be refined with acyclic subnets in N_1. For instance, subnet $N_1(\varphi^{-1}(p_1))$ is the refinement of place p_1 in N_2 shown in Fig. 8. Place refinements may lead to a split of transitions in N_2, e.g., transition t_1 in N_2 is split into two transitions t_{11} and t_{12} in N_1 (see Fig. 8).

Overall, an α-morphism is defined by a transition mapping between N_1 and N_2. If a transition in N_1 is mapped to a transition in N_2, their neighborhoods should also correspond. For example, in Fig. 8, transition t_{11} in N_1 is mapped to

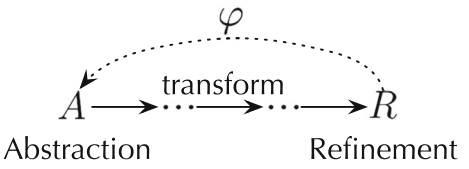


Fig. 9 Refinement through transformations

transition t_1 in N_2, and the neighborhood of t_{11} is mapped on the neighborhood of t_1. If a transition in N_1 is mapped to a place in N_2, then the neighborhood of this transition is also mapped to the same place. For example, in Fig. 8, transitions in subnet $N_1(\varphi^{-1}(p_1))$ are mapped to place p_1 in N_2, and the neighborhoods of these transitions are also mapped to place p_1. These and several other restrictions imposed on subnets in N_1 that can refine places in N_2 are discussed in [14]. These restrictions ensure the main motivation behind α-morphisms: behavioral properties valid for N_2 should also hold in its refinement N_1.

A systematic approach to defining α-morphisms was discussed in our earlier work [15]. The main idea of this approach is a step-wise application of local transformations to an abstract model to construct its refinement, as shown in Fig. 9 where each arc is a step of applying a structural transformation. These transformations are local since they affect only a part of a model, and the rest of the model remains untouched.

As proven in [15], every step of applying a transformation induces an α-morphism as well as their composition. Thus, after a series of transformations is applied, there will be an α-morphism from a transformed model towards the initial one. As shown in Fig. 9, there is an α-morphism $\varphi\colon R \to A$, and R is obtained by applying a series of transformations to A.

The main advantage to these transformations is the ability to redefine the notion of refinement without referring to the formal definition of α-morphisms since their direct application may be rather complicated.

A *transformation* is a tuple $\rho = (L, R, c_L, c_R)$ where:

1. L is the *left* part – a subnet to be transformed.
2. R is the *right* part – a subnet that replaces L.
3. c_L – constraints imposed on L.
4. c_R – constraints imposed on R.

Constraints c_L and c_R are structural, marking and transition labeling restrictions.

Let $N = (P, T, F, m_0, m_f)$ be a GWF-net. A transformation $\rho = (L, R, c_L, c_R)$ is *applicable* to N if there exists a subnet $N(X_L)$ corresponding to the left part of ρ. The application of ρ to N, schematically shown in Fig. 10, includes:

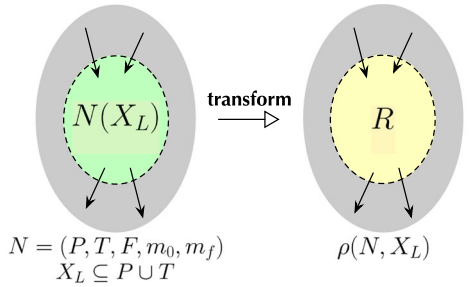


Fig. 10 Application of a transformation to N

Table 5 Refinement transformations

Transformation	Constraints
ρ_1: Place duplication	**1.** ${}^\bullet p_1 = {}^\bullet p = {}^\bullet p_2$; **2.** $p_1{}^\bullet = p^\bullet = p_2{}^\bullet$. **3.** $(p_1 \in m_0'$ and $p_2 \in m_0')$ iff $p \in m_0$.
ρ_2: Transition duplication	**1.** ${}^\bullet t_1 = {}^\bullet t = {}^\bullet t_2$; **2.** $t_1{}^\bullet = t^\bullet = t_2{}^\bullet$; **3.** t_1, t_2 have the same label as t.
ρ_3: Local transition introduction	**1.** ${}^\bullet t = \{p_1\}$, $t^\bullet = \{p_2\}$; **2.** $p_1{}^\bullet = {}^\bullet p_2 = \{t\}$; **3.** ${}^\bullet p_1 = {}^\bullet p$, $p_2{}^\bullet = p^\bullet$; **4.** t is not labeled with an interacting action.
ρ_4: Place split	**1.** $\lvert{}^\bullet p\rvert > 1$; **2.** ${}^\bullet p_1 \subset {}^\bullet p$, ${}^\bullet p_2 \subset p$; **3.** ${}^\bullet p_1 \cup {}^\bullet p_2 = {}^\bullet p$; **4.** ${}^\bullet p_1 \cap {}^\bullet p_2 = \varnothing$; **5.** $p_1{}^\bullet$, $p_2{}^\bullet$ are two complete copies of $p^\bullet$; **6.** ${}^\bullet(p_i{}^\bullet)\backslash\{p_i\} = {}^\bullet(p^\bullet)\backslash\{p\}$.

1. Identification and removal of $N(X_L)$ in N according to c_L.
2. Insertion of R to N connecting it with ${}^\bullet X_L{}^\bullet$ according to c_R.

The result of applying ρ to N is denoted by $\rho(N, X_L)$. We also write $N \xrightarrow{\rho} N'$ when $N' = \rho(N, X_L)$ and specification of an affected subnet is not important.

A set of structural transformations $RT = \{\rho_1, \rho_2, \rho_3, \rho_4\}$ used to refine GWF-nets is presented in Table 5. Corresponding constraints c_L and c_R are also given.

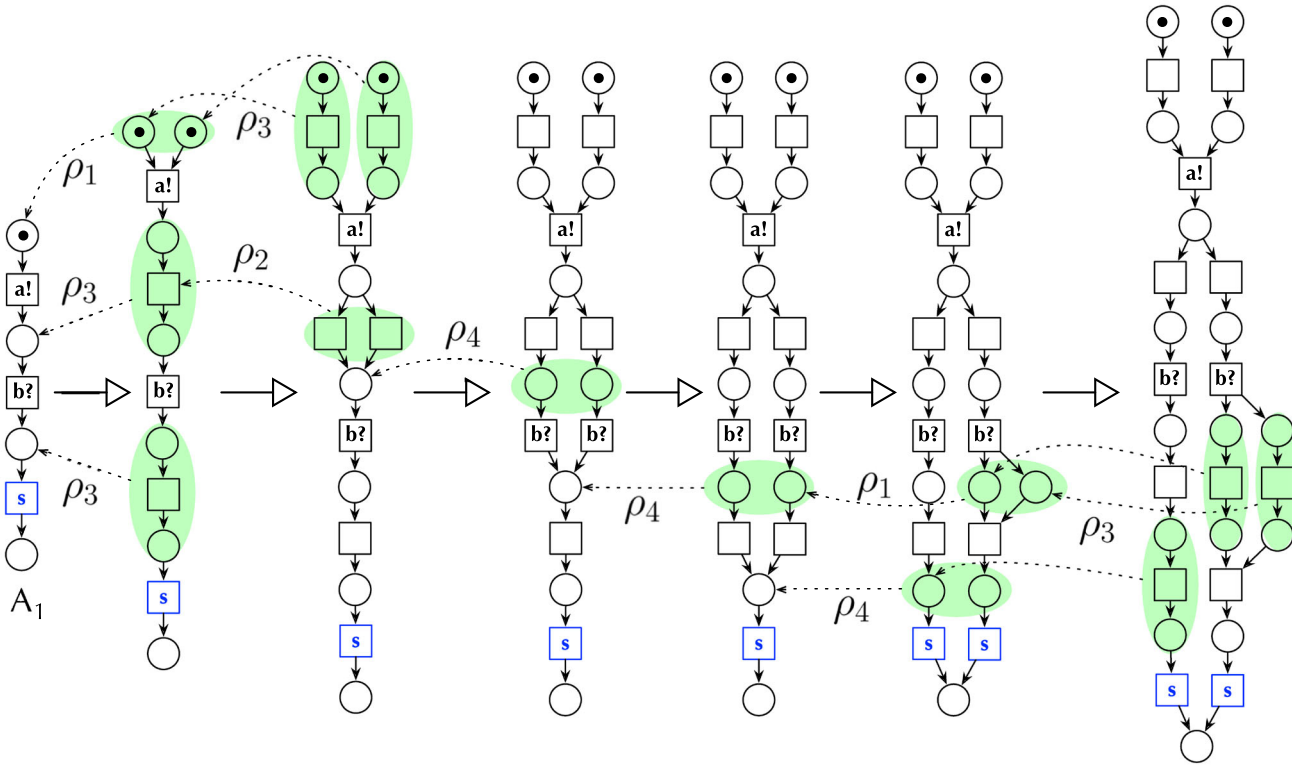


Fig. 11 Behavior of Agent 1 from Fig. 1 is a refinement of A_1 in interface pattern IP-10

Thus, we can finally redefine the notion of refinement through transformations.

Definition 7 (*Refinement*) Let N_1, N_2 be two GWF-nets. N_1 is a refinement of N_2 iff there exists a sequential application of transformations $\rho_1, \rho_2, ..., \rho_n \in RT$ leading from N_2 to N_1, i.e., $N_2 \xrightarrow{\rho_1} N_2' \xrightarrow{\rho_2} ... \xrightarrow{\rho_n} N_1$.

Correspondingly, the check ISREFINEMENT in Algorithm 1 comes down to the verification of the requirement imposed by Definition 7.

Let us consider the behavior of Agent 1 shown in Fig. 1. We show that it is a refinement of A_1 from pattern IP-10 (see Fig. 7(b)) since a subsequent application of transformations from the set RT exists, as provided in Fig. 11. Note that applications of transformations affecting disjoint subsets are shown as a single step. Overall, the sequence of transformations shown in Fig. 11 includes 13 steps.

A similar refinement check can be done for the behavior of Agent 2 from Fig. 1 to show that it is a refinement of A_2 from pattern IP-10. Thus, pattern IP-10 represents the interaction-oriented architecture viewpoints of the multi-agent system shown in Fig. 1.

4.3 The first correctness theorem

Here, we show that GWF-nets discovered using Algorithm 1 can replay all traces in these event logs. In other words, a GWF-net discovered from an event log L of a multi-agent system by Algorithm 1 *perfectly fits* L.

For what follows, L denotes an event log of a multi-agent system over $\Lambda = \Lambda_1 \cup \Lambda_2 \cup ... \cup \Lambda_k \cup In$.

We need to formalize a "*perfectly fits*" relation between a GWF-net and an event log.

Definition 8 (*Perfectly fits*) Let $N = (P, T, F, m_0, m_f)$ be a GWF-net with a transition labeling function $\ell: T \to \Lambda \cup \{\tau\}$. GWF-net N perfectly fits event log L iff $\forall \sigma \in L \, \exists w \in FS(N): \sigma = \ell(w)$.

We prove that a GWF-net S, discovered from an event log L of a multi-agent system using Algorithm 1, inherits the perfect fitness of an interface pattern and agent GWF-nets, discovered from agent log projections L_{Λ_i} with $i = 1, 2, ..., k$.

Theorem 1 *(Fitness preservation) Let* $IP = A_1 \circledast A_2 \circledast ... \circledast A_k$ *be an interface pattern with a transition labeling function* $\ell_{IP}: T_{IP} \to In \cup \{\tau\}$. *Let* R_i *be a refinement of* A_i *with a transition labeling function* $\ell_i: T_i \to \Lambda_i \cup \{\tau\}$ *for all* $i = 1, 2, ..., k$. *If* IP *perfectly fits* L_{In} *and* R_i *perfectly fits* L_{Λ_i} *for all* $i = 1, 2, ..., k$, *then* $S = R_1 \circledast R_2 \circledast ... \circledast R_k$ *with*

a transition labeling function $\ell\colon T \to \Lambda \cup \{\tau\}$ *perfectly fits* L.

Proof The proof is done by contradiction. Assume that S does not perfectly fit L. Then $\exists \sigma \in L$, s.t. $\nexists w \in FS(S)\colon \ell(w) = \sigma$. Since IP perfectly fits L_{In}, $\exists w_{IP} \in FS(IP)\colon \ell_{IP}(w_{IP}) = \sigma|_{In}$, because L_{In} is a log projection of L on $In \subseteq \Lambda$. Since R_i perfectly fits L_{Λ_i}, $\exists w_i \in FS(R_i) = \ell_i(w_i) = \sigma|_{\Lambda_i}$, because L_{Λ_i} is a log projection of L on $\Lambda_i \subseteq \Lambda$ for all $i = 1, 2, ..., k$. It is evident that w_{IP} and w_i (for all $i = 1, 2, ..., k$) are projections of a firing sequence $w' \in FS(S)$ on transitions labeled by In and Λ_i correspondingly. Since $\Lambda = \Lambda_1 \cup ... \cup \Lambda_k \cup In$ and taking the above into account, we have that $\ell(w') = \sigma$. It contradicts the assumption that $\nexists w \in FS(S)\colon \ell(w) = \sigma$. Hence S perfectly fits L. □

An immediate corollary of Theorem 1 gives the first correctness characteristic of Algorithm 1 as follows.

Corollary 1 *GWF-net S discovered from an event log L using Algorithm 1 perfectly fits L.*

4.4 The second correctness theorem

Using the formal framework behind the composition of GWF-nets, we prove that GWF-nets of multi-agent systems discovered by Algorithm 1 are sound. The main result of [9] is recalled in the following proposition.

Proposition 2 *(see* [9]*) Let* N_1, N_2, *and* R_1 *be three sound GWF-nets, s.t. there is an* α*-morphism* $\varphi\colon R_1 \to N_1$. *If* $N_1 \circledast N_2$ *is sound, then* $R_1 \circledast N_2$ *is sound.*

In other words, the soundness of a GWF-net composition $N_1 \circledast N_2$ is preserved if one of the two GWF-nets is replaced by its sound refinement.

Note that a refinement of a sound GWF-net, constructed by applying transformations (see Definition 7), is also a sound GWF-net. This follows from the fact that transformations do not introduce deadlocks [15].

Lemma 1 *Let* N_1, N_2 *be two GWF-nets, s.t.* N_1 *is a refinement of* N_2. *If* N_2 *is sound, then* N_1 *is sound.*

Then we prove that a GWF-net S, discovered from an event log L of a multi-agent system using Algorithm 1, preserves the soundness of an interface pattern and soundness of agent GWF-nets, discovered from log projections.

Theorem 2 *(Soundness preservation) Let* $IP = A_1 \circledast A_2 \circledast ... \circledast A_k$ *be an interface pattern, s.t.* A_i *is a sound GWF-net with* $i = 1, 2, ..., k$. *Let* R_i *be a refinement of* A_i *with* $i = 1, 2, ..., k$. *If* IP *is sound, then* $S = R_1 \circledast R_2 \circledast ... \circledast R_k$ *is also sound.*

Proof By Definition 7, since R_i is a refinement of A_i, there is a subsequent application of transformations $\rho_1, \rho_2, ..., \rho_n$ leading from A_i to R_i, i.e., $A_i \xrightarrow{\rho_1} ... \xrightarrow{\rho_n} R_i$, s.t. R_i is sound (by Lemma 1). Then there is an α-morphism $\varphi_i\colon R_i \to A_i$. The composition of GWF-nets is also a GWF-net. Let $IP' = A_1 \circledast ... \circledast A_{k-1}$, then $IP = IP' \circledast A_k$. By Proposition 2, since IP is sound and there is an α-morphism $\varphi_k\colon R_k \to A_k$, $IP' \circledast R_k$ is also sound. The composition of GWF-nets is commutative. Let $IP'' = A_1 \circledast ... \circledast A_{k-2} \circledast R_k$. Then $IP' \circledast R_k = IP'' \circledast A_{k-1}$. Since $IP'' \circledast A_{k-1}$ is sound and there is an α-morphism $\varphi_{k-1}\colon R_{k-1} \to A_{k-1}$, $IP'' \circledast R_{k-1}$ is also sound. Applying this reasoning further, we will arrive to a conclusion that $R_1 \circledast R_2 \circledast ... \circledast R_k$ is sound. □

An immediate corollary of Theorem 2 gives the second correctness characteristic of Algorithm 1 as follows.

Corollary 2 *GWF-net S discovered from an event log L using Algorithm 1 is sound.*

5 Experimental evaluation

In this section, we report the outcomes from a series of experiments conducted to evaluate the compositional process discovery algorithm. We used the twelve interface patterns considered in Sect. 3. According to the central hypothesis of our study, we analyze and compare the process models obtained using the ordinary direct process discovery with those discovered by our compositional approach.

5.1 Layout of experiments

Experiments were designed following the general scheme of the compositional process discovery (see Fig. 3) with two additional steps: *refinement* and *simulation*, as presented in Fig. 12. We introduced an *artificial source* of event logs, represented by a *reference* model. Reference GWF-nets are refinements of interface patterns. Thus, an event log obtained after simulating the behavior of a reference model will meet the assumptions of compositional process discovery. In addition, this approach conforms with a standard way of evaluating process discovery algorithms using so-called *artificial* event logs, which are supposed to have certain features.

Then, for every abstract interface pattern $IP = A_1 \circledast A_2 \circledast ... \circledast A_k$ from Sect. 3, the following procedure was executed.
Step 1. Construct a sound reference GWF-net $N_R = N_1 \circledast N_2 \circledast ... \circledast N_k$ where N_i is a refinement of A_i with $i = 1, 2, ..., k$.
Step 2. Simulate the behavior of N_R to obtain an event log L of a multi-agent system with k interacting agents over $\Lambda = \Lambda_1 \cup \Lambda_2 \cup ... \cup \Lambda_k$.
Step 3. Discover a sound GWF-net N_D *directly* from L.
Step 4. Construct k agent log projections $L_{\Lambda_1}, L_{\Lambda_2}, ..., L_{\Lambda_k}$ (by Definition 2).

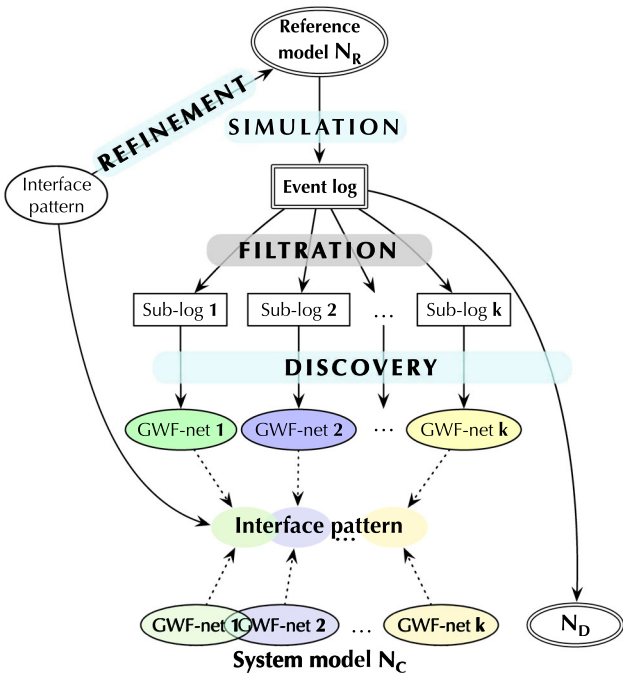


Fig. 12 Layout of experiments

Step 5. Discover k sound GWF-nets $N'_1, N'_2, ..., N'_k$ from agent log projections $L_{\Lambda_1}, L_{\Lambda_2}, ..., L_{\Lambda_k}$ respectively.
Step 6. Verify whether N'_i, discovered from L_{Λ_i}, is a refinement of A_i in IP with $i = 1, 2, ..., k$. If so, replace A_i with N'_i and construct $N_C = N'_1 \circledast N'_2 \circledast ... \circledast N'_k$.
Step 7. Compare the reference GWF-net N_R (Step 1), the directly discovered GWF-net N_D (Step 3), and the compositionally discovered GWF-net N_C (Step 6).

Below, we consider the main steps of our experiment plan in more detail.

Construction of a reference GWF-net (Step 1) and refinement verification (Step 6) are executed using Definition 7, based on the collection of structural GWF-net transformations, considered in Sect. 4.2. If N'_i, discovered from an agent log projection, is not a refinement of A_i in the interface pattern IP, we can consider how the pattern can be modified, preserving its soundness. Thus, we may extend Proposition 1.

Simulation of the reference GWF-net behavior (Step 2) is supported with an approach to event log generation introduced in [16]. This approach allows one to specify the behavior of agents and interface separately.

Step 3 and Step 5 are explicitly connected with the discovery of models from the event logs. The main requirement of a process discovery algorithm applied here is that it should produce sound models. Among the others, the *Inductive miner*, also mentioned earlier, always produces sound workflow nets.

Reference, directly and compositionally discovered GWF-nets are compared (Step 7) using standard and specifically developed *quality dimensions* that are described further.

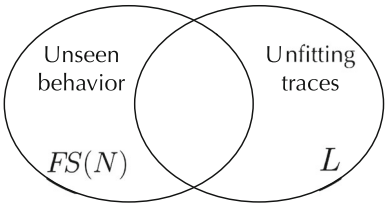


Fig. 13 Comparison between event log L and GWF-net N

The experimental results and corresponding conclusions are discussed in Sect. 5.3.

5.2 Conformance checking

Directly and compositionally discovered GWF-nets relate differently to the initial event log, obtained after simulating the behavior of the reference models. The general picture of relations between an event log L and a GWF-net N is given in Fig. 13.

Estimation of the correspondence between an event log and a process model is the main problem in the field of *conformance checking* [5]. In addition, within conformance checking, the structural complexity of process models is evaluated as well. There are four main quality dimensions offered in conformance checking: *fitness*, *precision*, *generalization* and *simplicity* [17]. They are aimed to build a holistic view of the quality of process models discovered from event logs.

In our experiments, we estimate *precision* and *simplicity* of the reference, directly and compositionally discovered GWF-nets with respect to the artificial event logs obtained after simulating the reference models, as specified by Step 7 in our experiment plan.

Fitness is a value in the interval [0, 1] that demonstrates how well a process model can replay every trace from an event log. In the general case shown in Fig. 13, a part of an event log (*unfitting traces*) may not be covered by the firing sequences in a process model. The more the number of unfitting traces in L is, the lower the fitness of a process model is. By Definition 8, a process model *perfectly fits* an event log (fitness = 1) if it can execute all traces in this event log, i.e., there are no unfitting traces.

Note that, by Corollary 1, GWF-nets obtained by Algorithm 1 perfectly fit event logs. Apart from that, existing process discovery algorithms allow configuring the desired fitness level. It may be necessary to decrease the fitness while working with *noisy* and real-life event logs, where there can be missing or duplicate actions, the wrong ordering of actions, etc. Artificial event logs do not have such problems. Thus, we do not need to estimate the fitness of reference, directly and compositionally discovered models.

Precision is a value in the interval [0, 1] that evaluates a ratio of the behavior allowed by a process model and not allowed by an event log (*unseen behavior* as shown in Fig. 13). Perfectly precise models can only replay the traces present in an event log. However, an event log represents only a finite fragment of all possible process executions. That is why perfectly precise models are of very restrictive use. A well-known approach, used in our experiments as well, to the precision estimation, is based on *aligning* the firing sequences of a process model with the traces in an event log [18].

The (structural) complexity of a discovered process model is captured by the *simplicity* dimension. We express the simplicity of a process model through:

- The number of places, transitions, and arcs;
- The number of *neighboring transitions* between pairs of different agents.

Recall that the compositional process discovery aims to build architecture-aware process models the structure of which indicates agent behavior and interactions. Thus, we expect the simplicity to be the main distinguishing feature of compositionally discovered GWF-nets compared to those discovered directly from event logs of multi-agent systems. Below, we explain the main idea behind the notion of neighboring transitions.

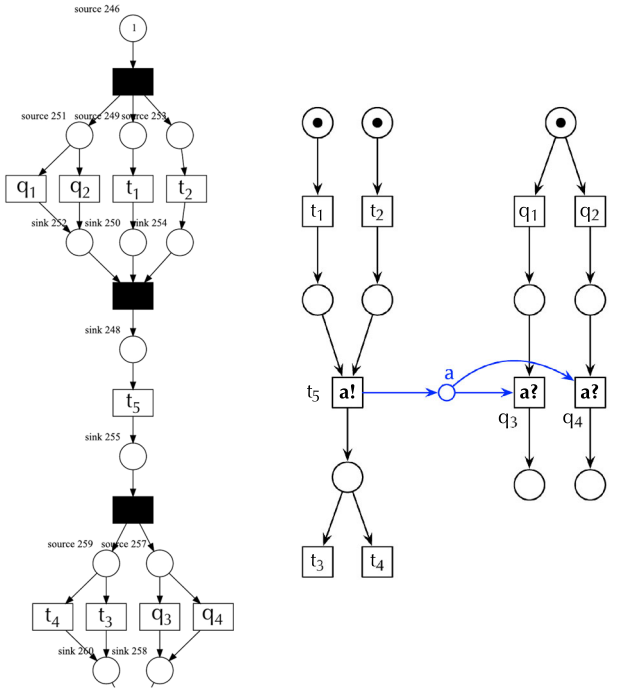
5.2.1 Neighboring transitions

The notion of neighboring transitions is introduced to estimate the extent to which a structure of a discovered process model covers the architecture viewpoints of a multi-agent system concerning agent interactions. In other words, an architecture-aware model of a multi-agent system explicitly shows the behavior of individual agents as well as the way they interact by exchanging messages and executing synchronous activities. Precise definitions are given below.

Let $L \in \mathcal{B}(\Lambda^+)$ be an event log of a multi-agent system over $\Lambda = \Lambda_1 \cup \Lambda_2 \cup ... \cup \Lambda_k$. Let $N = (P, T, F, m_0, m_f)$ be a GWF-net with a labeling function $\ell : T \to \Lambda \cup \{\tau\}$. According to ℓ, we can also partition T into k subsets corresponding to the behavior of different agents, i.e., $T = T_1 \cup T_2 \cup ... \cup T_k$, where transitions in T_i are labeled by actions from Λ_i with $i = 1, 2, ..., k$.

Transitions $t_i \in T_i$ and $t_j \in T_j$ such that $i \neq j$, $\ell(t_i) \neq \tau$, and $\ell(t_j) \neq \tau$ are called *neighboring* if there exists a path in N connecting t_i and t_j where the other transitions along this path are labeled by τ. Symbolically this is expressed as follows:

1. $(t_i, t_j) \in F^*$ where F^* is the reflexive transitive closure of F.
2. $\forall t \in T \setminus \{t_i, t_j\}$: if $(t_i, t) \in F^*$ and $(t, t_j) \in F^*$, then $\ell(t) = \tau$.



(a) $|\mathrm{NbT(N)}| = 4$ **(b)** $|\mathrm{NbT(N)}| = 2$

Fig. 14 Neighboring transitions

NbT(N) denotes the set of all neighboring transition pairs in N where symmetric pairs of neighboring transitions are counted as a single pair, i.e., $(t_1, t_2) \in \mathrm{NbT(N)} \Leftrightarrow (t_2, t_1) \notin \mathrm{NbT(N)}$. Intuitively, the bigger the $|\mathrm{NbT(N)}|$ is, the less transparent and understandable the structure of N is with respect to agent interactions. There are a lot of causally dependent transitions corresponding to the behavior of different agents. Below, we give an example of computing pairs of neighboring transitions.

Consider GWF-net fragments shown in Fig. 14. The first fragment (see Fig. 14(a)) is taken from the GWF-net shown in Fig. 2. The second fragment (see Fig. 14(b)) is taken from the GWF-net shown in Fig. 1.

Recall that the GWF-net shown in Fig. 2 were discovered directly from an event log of the GWF-net shown in Fig. 1. Both GWF-nets perfectly fit this event log. However, as mentioned in the Introduction, the direct discovery of a multi-agent system model may lead to inappropriate generalizations of agent behavior. For example, in the fragment shown in Fig. 14(b), transition q_3 can fire only after transition q_1, while in the fragment shown in Fig. 14(a), transition q_3 can fire after transitions q_1 or q_2.

These GWF-net fragments depict the behavior of a multi-agent system with two agents. The behavior of Agent 1 is

Table 6 Experimental results: asynchronous interface patterns

	IP-1	IP-2	IP-3	IP-4	IP-5	IP-6	IP-7	IP-8
Event logs								
Events	95052	149988	92668	102404	182452	123322	88068	157098
MIN trace	17	29	17	18	36	24	8	30
MEAN trace	19	30	19	20	36	25	18	31
MAX trace	21	31	20	23	37	25	29	32
Reference GWF-nets used for event log generation								
Places	35	52	45	41	59	60	31	70
Transitions	31	48	43	35	50	53	30	58
Arcs	73	110	96	86	121	131	78	148
NbT	4	4	2	6	7	7	12	8
Precision	0.73349	0.47346	0.77810	0.79798	0.37020	0.60955	0.82935	0.54382
GWF-nets discovered directly from event logs								
Places	52	74	83	56	88	97	30	107
Transitions	51	69	71	55	78	81	48	84
Arcs	126	182	178	134	210	214	104	230
NbT	48	124	46	48	81	74	69	117
Precision	0.75230	0.57704	0.89348	0.75958	0.48480	0.76818	0.32359	0.66462
Compositional process discovery								
Places	46	72	54	57	94	81	39	96
Transitions	41	71	51	49	80	71	38	78
Arcs	96	176	114	119	211	177	92	200
NbT	4	4	2	6	7	7	12	8
Precision	0.73639	0.43663	0.78777	0.80728	0.40350	0.62615	0.82180	0.56989

represented by transitions $\{t_1, t_2, t_3, t_4, t_5\}$. The behavior of Agent 2 is represented by transitions $\{q_1, q_2, q_3, q_4\}$. The first fragment shown in Fig. 14(a) has four pairs of neighboring transitions, i.e., $\{(q_1, t_5), (q_2, t_5), (t_5, q_3), (t_5, q_4)\}$. However, in the second fragment shown in Fig. 14(b), there are only two pairs of neighboring transitions, i.e., $\{(t_5, q_3), (t_5, q_4)\}$, exactly corresponding to the transitions through which Agents 1 and 2 interact.

Further computation of neighboring transition pairs in the GWF-nets from Figs. 1 and 2 will lead to the following observations:

1. In a GWF-net discovered by Algorithm 1, the number of neighboring transitions directly corresponds to the interacting transitions.
2. In a GWF-net discovered directly from an event log of a multi-agent system, there are far more pairs of neighboring transitions since transitions corresponding to different agents are tightly connected.

5.3 Experimental results

Table 6 presents precision and simplicity evaluations of the reference, directly and compositionally discovered GWF-nets of multi-agent systems where agent interactions are specified by asynchronous interface patterns IP-1, IP-2, ..., IP-8.

Table 7 presents precision and simplicity evaluations of the reference, directly and compositionally discovered GWF-nets of multi-agent systems whose architecture is described by mixed asynchronous-synchronous interface patterns IP-9, IP-10, IP-11, and IP-12.

In these two tables, we also provide the information on artificial event logs obtained by simulating the reference GWF-nets, including the total number of events together with the minimum, maximum, and average trace length in these event logs. The longest traces are represented in the event log of pattern IP-5 since there are parallel branches in the behavior of interacting agents. The most notable difference between the minimum and maximum trace lengths is represented in the event log of pattern IP-7 since there are loops in the behavior of interacting agents.

For a better interpretation of the experimental results, Table 8 provides pair-wise comparison between the precision and simplicity evaluations given in Table 6 and in Table 7, where we have computed the percentage change in the characteristics of:

– Directly discovered and reference GWF-nets;
– Compositionally discovered and reference GWF-nets;

Table 7 Experimental results: mixed interface patterns

	IP-9	IP-10	IP-11	IP-12
Event logs				
Events	115000	102548	160000	88089
MIN trace	23	20	32	17
MEAN trace	23	21	32	18
MAX trace	23	21	32	18
Reference GWF-nets used for event log generation				
Places	53	41	50	44
Transitions	52	37	44	42
Arcs	128	89	103	97
NbT	6	6	6	3
Precision	0.76541	0.83729	0.43610	0.77731
GWF-nets discovered directly from event logs				
Places	68	49	84	73
Transitions	79	53	80	67
Arcs	186	126	202	168
NbT	75	47	71	41
Precision	0.60369	0.69177	0.45910	0.80212
Compositional process discovery				
Places	66	49	69	55
Transitions	64	45	63	52
Arcs	155	109	155	121
NbT	6	6	6	3
Precision	0.76785	0.81475	0.46640	0.77522

– Compositionally and directly discovered GWF-nets.

Based on these pair-wise comparison results, we report the main conclusions and outcomes from the experiments on evaluating the compositional process discovery approach.

We start with the analysis of the simplicity comparisons given in Table 8.

An increase in the number of nodes in the directly and compositionally discovered GWF-nets, next to the reference GWF-nets, is mainly caused by additional τ-transitions. They connect the standard behavioral constructions such as the sequential, concurrent, or alternative control-flows of actions executed by different agents. Conversely, the compositional process discovery shows an overall decrease or a moderate increase in the number of nodes compared with the direct process discovery since we separate the behavior of different agents. The separation of agent behavior is also justified by the changes in the number of neighboring transitions. One may observe a multiplicative increase in the number of the neighboring transitions in the directly discovered GWF-nets. The compositionally discovered GWF-nets have the same number of the neighboring transitions as the reference GWF-nets. These transitions correspond exactly to actions through which agents interact, while the rest of agent behavior is independent since it is not involved in their interactions.

We next consider the precision comparisons also provided in Table 8.

Most directly discovered GWF-nets improve the precision since they are far more oriented to the corresponding event logs. The precision of the reference and compositionally discovered GWF-nets are lower next to the directly discovered GWF-nets since the separation of agent behavior leads to a corresponding increase in the amount of unseen behavior, as shown in Fig. 13. This precision decrease is a payment for making process models of multi-agent systems architecture-aware. However, in the case of the interface patterns with complicated and mixed agent interactions, namely IP-7, IP-9, ..., and IP-12, we observe a decrease or a negligible increase in precision. The inappropriate generalizations of agent behavior were the main reason for this precision decrease. In conclusion to the precision comparative analysis, we also see that the compositionally discovered GWF-nets preserve almost the same precision level next to the reference GWF-nets since changes in the corresponding values are less than 10%.

To sum up the discussion of the experimental results, we take a closer look at the outcomes reported for interface patterns IP-2 and IP-7. Following the steps of our experiment plan for these interface patterns, we encountered the following issues:

– Interface pattern inconsistencies (IP-2);
– The most notable decrease in the precision (IP-7).

We further discuss the reasons for these problems.

5.3.1 Pattern inconsistencies: the case of IP-2

The experiment with the asynchronous interface pattern IP-2 shown in Fig. 6(b) led to the following problem. Agent GWF-nets, N_1 and N_2, discovered from log projections, were not the proper refinements of A_1 and A_2 in IP-2, according to the requirement of Definition 7. Thus, the corresponding IsRefinement test in Algorithm 1 was not passed.

As mentioned in Sect. 5.1, in the detailed description of the experiment plan, we would try to reconfigure an interface pattern in this case. Then, we determined that there exist two sequences of refinement transformations (see Table 5) that lead from A'_1 and A'_2 shown in Fig. 15 to agent GWF-nets N_1 and N_2.

Intuitively, it can be seen that the two pairs of concurrent actions, "a!", "b!" and "a?", "b?", were discovered as sequential actions respectively. The main reason for this is the lack of different process executions in an event log generated by the reference GWF-net.

Table 8 Experimental results: changes in simplicity and precision evaluations

	IP-1	IP-2	IP-3	IP-4	IP-5	IP-6	IP-7	IP-8	IP-9	IP-10	IP-11	IP-12
Directly discovered GWF-nets compared to reference GWF-nets												
Places	+49%	+42%	+84%	+37%	+49%	+62%	−3%	+53%	+28%	+20%	+68%	+66%
Transitions	+65%	+44%	+65%	+57%	+56%	+53%	+60%	+45%	+52%	+43%	+82%	+60%
Arcs	+72%	+66%	+85%	+56%	+74%	+63%	+33%	+55%	+45%	+42%	+96%	+73%
NbT	×12	×31	×23	×12	×11.6	×10.6	×5.6	×14.6	×12.5	×7.8	×11.8	×13.7
Precision	+3%	+22%	+15%	-5%	+31%	+26%	−61%	+22%	−21%	−17%	+5%	+3%
Compositionally discovered GWF-nets compared to reference GWF-nets												
Places	+31%	+40%	+20%	+39%	+59%	+35%	+26%	+37%	+25%	+20%	+38%	+25%
Transitions	+32%	+48%	+19%	+40%	+60%	+34%	+27%	+35%	+23%	+22%	+43%	+24%
Arcs	+32%	+60%	+19%	+38%	+74%	+35%	+18%	+35%	+21%	+23%	+51%	+25%
NbT	coincides with the values of reference GWF-nets											
Precision	+0,4%	-8%	+1%	+1%	+9%	+3%	−1%	+5%	+0.3%	−3%	+7%	-0.3%
Compositionally discovered GWF-nets compared to directly discovered GWF-nets												
Places	−12%	−3%	-35%	+2%	+7%	−17%	+30%	−10%	-3%	0%	−18%	-25%
Transitions	−20%	+3%	−28%	−11%	+3%	−12%	−21%	−7%	−19%	−15%	−21%	−22%
Arcs	−24%	−3%	−36%	−11%	+1%	−17%	−12%	−13%	−17%	−14%	−23%	−28%
NbT	−92%	−97%	−96%	−88%	−91%	−91%	−83%	−94%	−92%	−87%	−92%	−93%
Precision	−2%	−24%	−12%	+6%	−17%	−19%	×2.5	−14%	+27%	+18%	+2%	−3%

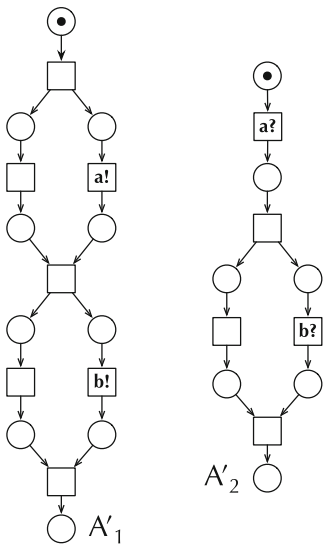


Fig. 15 Modifications of A_1 and A_2 in IP-2

Having considered $A'_1 \circledast A'_2$ as the new interface pattern and verified its soundness (see Proposition 1), we actually experimented with the modified version of the interface pattern IP-2.

5.3.2 Precision drop: the case of IP-7

The experiment with the multiple transmission interface pattern IP-7, shown in Fig. 6(g), also deserves to be highlighted. The directly discovered GWF-net exhibits a sharp decrease in its precision compared to the reference and compositionally discovered GWF-nets.

Figure 16 shows the GWF-net discovered directly from an event log generated by the reference GWF-net of pattern IP-7. As seen from this GWF-net, its structure contains several joint blocks of actions executed by different agents. The structure of this model does not cover the interaction-oriented architecture viewpoints of a system with two interacting agents exchanging messages until one of them decides to stop the exchange.

Thus, the complicated nature of agent interactions can hardly be reconstructed directly from an event log of a multi-agent system. The identification of agent behavior and the interface pattern refinement check allows us to decompose this problem and improve the quality of a multi-agent system model.

5.4 Technical support of experiments

According to the plan discussed in Sect. 5.1, experiments were conducted using a PC with the following characteristics:

1. CPU Intel Core i7 3.70GHz.
2. 32 Gb RAM.
3. 64-bit OS Windows 10 Pro.

The generation and filtration of artificial event logs, the discovery of agent GWF-nets, and precision evaluation were

supported by the *ProM* software [19]. This is the plugin-extendable tool, where various process discovery algorithms are implemented.

The experimental data, including the artificial event logs (XES-files), reference, directly, and compositionally discovered GWF-nets (PNML-files), are accessible via the open Zenodo repository [20].

5.5 Limitations of interface patterns

Interface patterns play an essential role in proving the correctness of the compositional process discovery algorithm (see Corollary 1 and 2). The experimental results found that the agent interaction requirements imposed by interface patterns might not be fully satisfied by agent GWF-nets discovered from filtered sub-logs. The main reason for this problem is the incompleteness of event logs. They represent only a "finite snapshot" of all possible executions generated by concurrent interactions among agents in multi-agent systems. To tackle the problem, one should either correspondingly adapt an interface pattern verifying its soundness, as exemplified in Sect. 5.3.1, or process event logs with a bigger number of different trace classes.

Another limitation is the manual selection of an interface pattern and the manual construction of refinement transformation sequences using Definition 7. Manual work can result in the wrong pattern choice and mistakes in checking whether an agent GWF-net is a refinement of the respective subnet in an interface pattern. We plan to overcome these issues by developing an algorithm for the refinement check IsRefinement (see Algorithm 1) and by extending the collection of typical and sound interface patterns.

6 Related works

As mentioned in the Introduction, there are many algorithms for the automated discovery of process models from event logs. Among the most widespread ones are the following: Inductive miner [13], Fuzzy miner [21], Heuristic miner [22], ILP-based (integer linear programming) miner [23], Region Theory-based miner [24], and Genetic miner [25]. A. Augusto et al. [2] conducted a comprehensive and systematic review of these and other process discovery algorithms. The existing process discovery algorithms can tackle different problems connected with the representation of event data in logs. They include noise, e.g., the wrong ordering of logged actions, duplicate or missed actions, and incompleteness, i.e., an event log represents only a finite fragment of all possible execution sequences.

In general, the synthesis of Petri nets from low-level behavioral representations, e.g., transition systems, is a well-known problem that is to decide whether a given transition

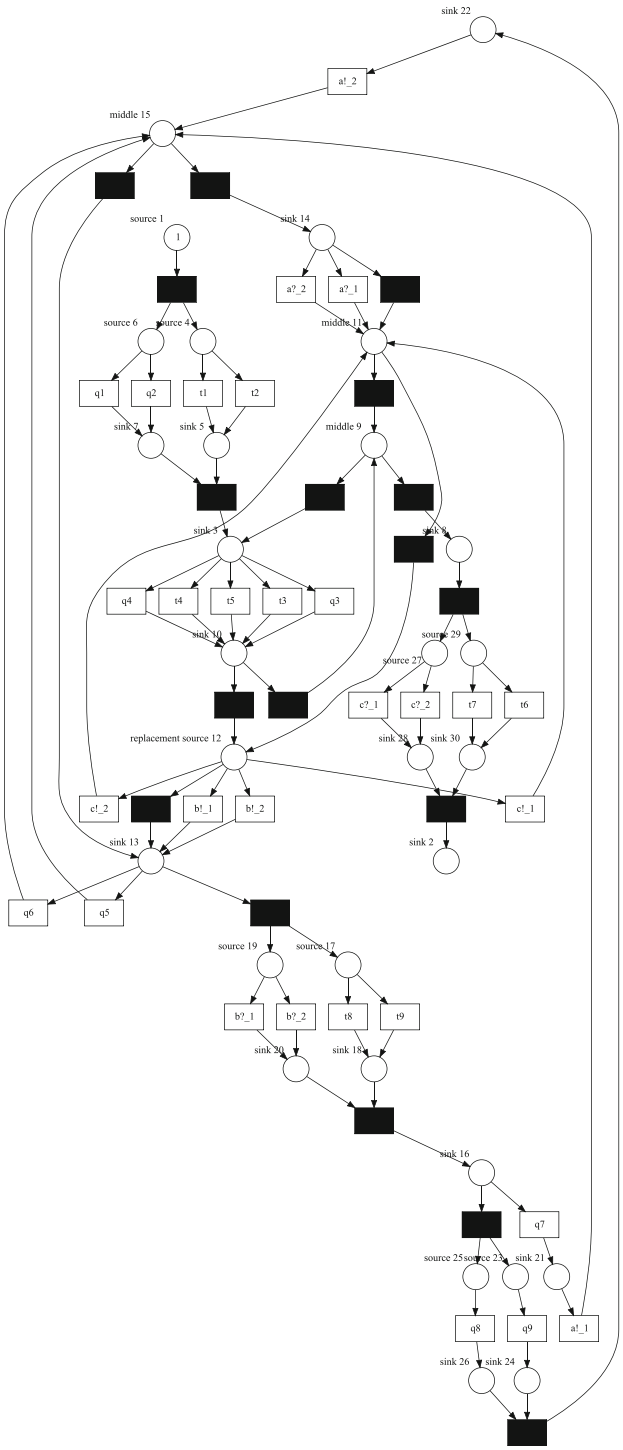


Fig. 16 Directly discovered GWF-net: pattern IP-7

system is isomorphic to the reachability graph of some Petri net and then to construct this net [26]. Region theory [27] is the main formal tool used to solve the Petri net synthesis problem. It undoubtedly found use in process discovery when an event log is represented by a finite transition system.

The experimental part of our study is supported with the Inductive miner since it discovers a process model with

necessary features, including soundness and state machine decomposability. We incorporated this algorithm into the compositional process discovery.

Conformance checking [5] is an essential part of process mining along with process discovery. It is aimed to assess the quality of process models discovered from event logs since different algorithms yield different process models, and they are to be compared. The four main quality dimensions in process discovery are fitness, precision, generalization, and simplicity. J. Buijs et al. [17] discussed the role of these dimensions. The review [2] also indicated the lack of universal measures of the fitness and precision applicable to a wide range of process discovery algorithms. W. van der Aalst [28] discussed the same question from a slightly different perspective, focusing on the urgent need for the consistent requirements of quality measures since there is a significant increase in the use of process discovery algorithms in commercial software tools.

Within the framework of this paper, our focus was on distinguishing process models of multi-agent systems discovered by the compositional approach. In this light, we measured the structural complexity of agent interactions by assessing the number of neighboring transitions corresponding to the behavior of different agents.

The problem of discovering structured process models from event logs is not entirely new. Researchers studied this problem in several contexts. In general, there exists the continuum of process models: from *Spaghetti* (poorly structured models, e.g., the one presented in Fig. 16 to *Lasagna* (models with a clear structure). Subtle differences between well-structured, structured, and semi-structured process models are hard to formalize [1]. For example, process models discovered by the Inductive miner are called *well-structured* since they are recursively constructed from building blocks. They correspond to the basic constructs such as sequential, parallel, alternative, and cyclic executions.

Researchers offer different techniques to improve the structure of discovered models, e.g., in [29], and to produce already well-structured process models [30–32]. Compositional approaches to improving the structure of discovered process models were proposed as well. A. Kalenkova et al. [33] showed how to discover a readable model from an event log by decomposing the extracted transition system. A. Kalenkova and I. Lomazova [34] studied an advanced technique to deal with cancellations — "exceptional" behaviors — in the process execution and to produce clear and structured process models. In addition, W. van der Aalst et al. [35] proposed an approach for the compositional process discovery based on localizing events using region theory to improve the overall quality of discovered process models. A method for compositional modeling and discovery of structured object-centric Petri nets was proposed by W. van der Aalst and A. Berti in [36], where they used special transition fusions. M. Stierle et al. [37] discussed some design principles of discovering comprehensible models from event logs. They defined metrics estimating the extent to which a discovered model meets these principles. In our study, the clear architecture-aware structure of multi-agent system models results from the independent discovery of process models for interacting agents from log projections.

A large amount of literature is devoted to Petri net composition, e.g., general compositional frameworks and approaches are discussed in [3,38,39]. The main problem of a compositional approach, also considered in this paper, is to preserve component properties in their composition. The incorrect composition can be caused by the poor specifications of component interactions resulting in the violation of component behavioral properties. L. de Alfaro and T. Henziger [12,40] emphasized the importance of the proper interface specification in the component-based design and development of complex software systems. Moreover, according to these works, interfaces should describe component interactions at the most abstract level without exposing their internal behavior.

We used morphisms on Petri nets to achieve the inheritance of behavioral component properties. Morphisms provide a convenient formal tool from category theory, used for the modular design of information systems with interacting components. They give a natural and rigid framework to study the properties of different Petri net compositions. These properties may include the preservation of firing sequences, place/transition invariants, and others. Petri net composition based on morphisms was discussed in several works, including [41–44], with different aspects of preserving behavioral and structural properties.

For safe net systems, used in our paper to model the behavior of interacting agents and multi-agent systems, L. Bernardinello et al. [14] proposed a class of α-morphisms. They support the refinement of places to identify whether agent models are proper refinements of the corresponding parts in interface patterns. Using α-morphisms also accounts for preserving the soundness of agents in a complete model of a multi-agent system. However, we did not apply α-morphisms directly since the required theoretical background is rather extensive. Instead, we redefined the notion of refinement via the local structural transformations proposed in our earlier work [15]. These transformations are the basis of a systematic approach to the definition of α-morphisms.

Within the compositional approach to discovering process models of multi-agent systems, we assume that experts provide specifications of agent interactions in advance. Identifying an interface model from a raw event log of a multi-agent system is another task that is out of the scope of this paper. We designed a collection of specific interface patterns using typical service interaction patterns studied by A. Barros et al. in [8]. They provide generic solutions

to the specification of complex component interactions in large-scale systems. G. Decker et al. [45] and D. Campagna et al. [46] discussed the practical application of interaction patterns to construct corresponding BPMN process models. The correctness of interface patterns was also studied by G. Decker et al. in [47] and by W. van der Aalst et al. in [48]. They formalized patterns using process algebras and open Petri nets — a class of Petri nets with distinguished input and output places. The authors used operating guidelines to construct services correctly interacting with the given one. In our case, an interface pattern comprises highly abstract representations of all interacting agents. Moreover, since interface patterns are known to be correct, they can be reused for all properly constructed refinements, representing the concrete behavior of agents.

This paper is based on our previous works [49,50]. We extended the earlier achieved results by considering the formalization of multilateral interactions (pattern IP-8) and the mixed interaction patterns (patterns IP-9, IP-10, IP-11, and IP-12). We relied on the mathematical framework for the associative composition of generalized workflow nets studied in [9].

7 Conclusions and future work

This paper proposed a *compositional* approach to discovering *architecture-aware* and *sound* process models from event logs generated by multi-agent systems. The structure of an architecture-aware model is self-explanatory, i.e., it explicitly shows the behavior of individual agents and their interactions. Our solution involves an additional interface model and includes three steps. Firstly, we filter event logs by actions belonging to each agent and obtain a set of sub-logs. Secondly, agent models are discovered from these sub-logs with the help of an existing process discovery algorithm. Finally, we check whether there is a mapping of agent models to the corresponding parts in an interface.

Arbitrary interfaces are not considered since it is easy to arrange agent interactions leading to a deadlock. We designed a collection of specific *interface patterns* describing typical agent interactions. An interface pattern describes the architecture of a multi-agent system. Moreover, agent interactions specified by an interface pattern will not violate the soundness of agent behavior. The set of presented interface patterns is based on service interaction patterns studied earlier. It can also be extended with new models of agent interactions, provided that each new pattern is sound. If a map from agent models towards the corresponding parts in an interface pattern exists, then we can replace this part in a pattern with a discovered agent model. As a result, we obtain an architecture-aware model of a multi-agent system if all agent models are successfully mapped on an interface pattern. Otherwise, when only some agent models can be mapped on an interface pattern, we construct an approximation of a multi-agent system model we are looking for. In this case, an interface pattern needs to be modified, such that all agent models can be mapped on it.

The mathematical framework for finding a mapping of agent models to an interface pattern is based on structural transformations inducing α-morphisms. We formally demonstrated the correctness of the compositional process discovery from two perspectives. Firstly, a multi-agent system model perfectly fits an event log, i.e., for all traces in this event logs, there exists an execution in the model. Secondly, a multi-agent system model inherits the soundness of an interface pattern and individual agent models.

To evaluate the proposed compositional approach, we conducted a series of experiments. We compared the quality of the process models discovered directly from the artificial event logs of multi-agent systems with the quality of the process models discovered using the interface patterns. The experimental results confirm the overall improvement in the structure of architecture-aware process models of multi-agent systems since agent behavior, not involved in their interactions, is structurally separated.

The proposed approach is applicable to distributed systems with components that can be represented as business processes. The control-flow of these processes can be formalized using generalized workflow nets, making the soundness property relevant for the analysis. The main limitation of the compositional process discovery is the manual selection of interface patterns according to information provided by experts. This can result in the further adaptation and soundness verification of modified interface patterns. However, the number of interacting actions is usually significantly less than the number of local actions of agents.

As for future research, we plan to continue our work in several directions that are also focused on overcoming the limitations. Firstly, the application of α-morphisms and the corresponding structural transformations does not allow refining acyclic interface patterns with cyclic behavior. We want to consider possible constraint relaxations, such that the overall correctness is preserved. Then the applicability of interface patterns will be extended. Secondly, we plan to augment the presented collection of interface patterns with new interaction models, especially considering the broadcast communication, and apply the compositional approach to real-life examples of event logs. Finally, we also plan to work on an approach to identifying interfaces from event logs of multi-agent systems.

We are grateful to the reviewers for their valuable suggestions that helped us to improve the presentation of the main contributions of our study.

References

1. van der Aalst, W.: Process Mining: Data Science in Action. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49851-4
2. Augusto, A., Conforti, R., Dumas, M., Rosa, M., Maggi, F., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: review and benchmark. IEEE Trans. Knowl. Data Eng. **31**(4), 686–705 (2019). https://doi.org/10.1109/TKDE.2018.2841877
3. Reisig, W.: Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-33278-4
4. Kalenkova, A., van der Aalst, W., Lomazova, I., Rubin, V.: Process mining using BPMN: relating event logs and process models. Softw. Syst. Model. **16**, 1019–1048 (2017). https://doi.org/10.1007/s10270-015-0502-0
5. Carmona, J., van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking: Relating Processes and Models. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99414-7
6. van der Aalst, W.: Workflow verification: finding control-flow errors using petri-net-based techniques. In: van der Aalst, W., Desel, J., Oberweis, A. (eds.) Business Process Management: Models, Techniques, and Empirical Studies. Lecture Notes in Computer Science, vol. 1806, pp. 161–183. Springer, Heidelberg (2000). https://doi.org/10.1007/3-540-45594-9_11
7. van der Aalst, W., van Hee, K., ter Hofstede, A., Sidorova, N., Verbeek, H., Voorhoeve, M., Wynn, M.: Soundness of workflow nets: classification, decidability, and analysis. Form. Asp. Comput. **23**, 333–363 (2011). https://doi.org/10.1007/s00165-010-0161-4
8. Barros, A., Dumas, M., ter Hofstede, A.: Service interaction patterns. In: van der Aalst, W., Benatallah, B., Casati, F., Curbera, F. (eds.) Business Process Management. Lecture Notes in Computer Science, vol. 3649, pp. 302–318. Springer, Heidelberg (2005). https://doi.org/10.1007/11538394_20
9. Bernardinello, L., Lomazova, I., Nesterov, R., Pomello, L.: Soundness-preserving composition of synchronously and asynchronously interacting workflow net components (2020). https://arxiv.org/pdf/2001.08064.pdf
10. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, Boston (1994)
11. van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow patterns. Distrib. Parallel Databases **14**, 5–51 (2003). https://doi.org/10.1023/A:1022883727209
12. de Alfaro, L., Henzinger, T.: Interface-based design. In: Broy, M., Grünbauer, J., Harel, D., Hoare, T. (eds.) Engineering Theories of Software Intensive Systems, pp. 83–104. Springer, Dordrecht (2005). https://doi.org/10.1007/1-4020-3532-2_3
13. Leemans, S., Fahland, D., van der Aalst, W.: Discovering block-structured process models from event logs-a constructive approach. In: Colom, J., Desel, J. (eds.) Application and Theory of Petri Nets and Concurrency. Lecture Notes in Computer Science, vol. 7927, pp. 311–329. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38697-8_17
14. Bernardinello, L., Mangioni, E., Pomello, L.: Local state refinement and composition of elementary net systems: an approach based on morphisms. In: Koutny, M., van der Aalst, W., Yakovlev, A. (eds.) Transactions on Petri Nets and Other Models of Concurrency VIII. Lecture Notes in Computer Science, vol. 8100, pp. 48–70. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40465-8_3
15. Bernardinello, L., Lomazova, I., Nesterov, R., Pomello, L.: Property-preserving transformations of elementary net systems based on morphisms. In: Transactions on Petri Nets and Other Models of Concurrency XVI (ToPNoC). Springer (2022, to appear)
16. Nesterov, R., Mitsyuk, A., Lomazova, I.: Simulating behavior of multi-agent systems with acyclic interactions of agents. Proceed. Inst. Syst. Program. RAS **30**(3), 285–302 (2018). https://doi.org/10.15514/ISPRAS-2018-30(3)-20
17. Buijs, J., van Dongen, B., van der Aalst, W.: On the role of fitness, precision, generalization and simplicity in process discovery. In: On the Move to Meaningful Internet Systems: OTM 2012. Lecture Notes in Computer Science, vol. 7565, pp. 305–322. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33606-5_19
18. Adriansyah, A.: Aligning observed and modeled behavior. Ph.D. thesis, Mathematics and Computer Science (2014). https://doi.org/10.6100/IR770080
19. van Dongen, B., de Medeiros, A., Verbeek, H., Weijters, A., van der Aalst, W.: The ProM framework: a new era in process mining tool support. In: Ciardo, G., Darondeau, P. (eds.) Applications and Theory of Petri Nets 2005. Lecture Notes in Computer Science, vol. 3536, pp. 444–454. Springer, Heidelberg (2005). https://doi.org/10.1007/11494744_25
20. Nesterov, R.: Compositional discovery of architecture-aware and sound process models from event logs of multi-agent systems: experimental data. (Version 1) [Data set]. Zenodo (2021). https://doi.org/10.5281/zenodo.5830863
21. Günther, C., van der Aalst, W.: Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) Business Process Management. Lecture Notes in Computer Science, vol. 4714, pp. 328–343. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_24
22. Weijters, A., Ribeiro, J.: Flexible heuristics miner (FHM). In: 2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), pp. 310–317. IEEE (2011). https://doi.org/10.1109/CIDM.2011.5949453
23. van der Werf, J., van Dongen, B., Hurkens, C., Serebrenik, A.: Process discovery using integer linear programming. Fundam. Inform. **94**(3–4), 387–412 (2009). https://doi.org/10.3233/FI-2009-136
24. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process mining based on regions of languages. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) Business Process Management. Lecture Notes in Computer Science, vol. 4714, pp. 375–383. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75183-0_27
25. van der Aalst, W., de Medeiros, A., Weijters, A.: Genetic process mining. In: Ciardo, G., Darondeau, P. (eds.) Applications and Theory of Petri Nets. Lecture Notes in Computer Science, vol. 3536, pp. 48–69. Springer, Heidelberg (2005). https://doi.org/10.1007/11494744_5
26. Badouel, E., Bernardinello, L., Darondeau, P.: Petri Net Synthesis. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47967-4
27. Badouel, E., Darondeau, P.: Theory of regions. In: Reisig, W., Rozenberg, G. (eds.) Lectures on Petri Nets I: Basic Models: Advances in Petri Nets. Lecture Notes in Computer Science, vol. 1491, pp. 529–586. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-65306-6_22
28. van der Aalst, W.: Relating process models and event logs – 21 conformance propositions. In: Proceedings of the International Workshop on Algorithms and Theories for the Analysis of Event Data 2018, CEUR Workshop Proceedings, vol. 2115, pp. 56–74. CEUR-WS.org (2018)
29. van der Aalst, W., Gunther, C.: Finding structure in unstructured processes: The case for process mining. In: Seventh International Conference on Application of Concurrency to System Design (ACSD 2007), pp. 3–12. IEEE (2007). https://doi.org/10.1109/ACSD.2007.50

30. Buijs, J.: Flexible evolutionary algorithms for mining structured process models. Ph.D. thesis, Eindhoven University of Technology (2014)
31. De Smedt, J., De Weerdt, J., Vanthienen, J.: Multi-paradigm process mining: retrieving better models by combining rules and sequences. In: On the Move to Meaningful Internet Systems: OTM 2014 Conferences. Lecture Notes in Computer Science, vol. 8841, pp. 446–453. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-45563-0_26
32. de San Pedro, J., Cortadella, J.: Mining structured petri nets for the visualization of process behavior. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, p. 839–846. ACM (2016). https://doi.org/10.1145/2851613.2851645
33. Kalenkova, A., Lomazova, I., van der Aalst, W.: Process model discovery: a method based on transition system decomposition. In: Ciardo, G., Kindler, E. (eds.) Application and Theory of Petri Nets and Concurrency. Lecture Notes in Computer Science, vol. 8489, pp. 71–90. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07734-5_5
34. Kalenkova, A., Lomazova, I.: Discovery of cancellation regions within process mining techniques. Fundam. Inform. **133**, 197–209 (2014). https://doi.org/10.3233/FI-2014-1071
35. van der Aalst, W., Kalenkova, A., Rubin, V., Verbeek, E.: Process discovery using localized events. In: Devillers, R., Valmari, A. (eds.) Application and Theory of Petri Nets and Concurrency. Lecture Notes in Computer Science, vol. 9115, pp. 287–308. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19488-2_15
36. van der Aalst, W., Berti, A.: Discovering object-centric petri nets. Fundam. Inform. **175**, 1–40 (2020). https://doi.org/10.3233/FI-2020-1946
37. Stierle, M., Zilke, S., Dunzer, S., Tenscher, J., Karagegova, G.: Design principles for comprehensible process discovery in process mining. In: ECIS 2020 Proceedings. Research Papers. AIS eLibrary (2020)
38. Best, E., Devillers, R., Koutny, M.: Petri Net Algebra. In: Brauer, W., Rozenberg, G., Salomaa, A. (eds.) Monographs in Theoretical Computer Science. An EATCS Series. Springer, Heidelberg (2001). https://doi.org/10.1007/978-3-662-04457-5
39. Girault, C., Rüdiger, V.: Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-662-05324-9
40. de Alfaro, L., Henzinger, T.: Interface theories for component-based design. In: Henzinger, T., Kirsch, C. (eds.) Embedded Software. Lecture Notes in Computer Science, vol. 2211, pp. 148–165. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45449-7_11
41. Bednarczyk, M., Bernardinello, L., Caillaud, B., Pawłowski, W., Pomello, L.: Modular system development with pullbacks. In: van der Aalst, W., Best, E. (eds.) Applications and Theory of Petri Nets. Lecture Notes in Computer Science, vol. 2679, pp. 140–160. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44919-1_12
42. Bernardinello, L., Monticelli, E., Pomello, L.: On preserving structural and behavioral properties by composing net systems on interfaces. Fundam. Inform. **80**(1–3), 31–47 (2007)
43. Padberg, J., Urbášek, M.: Rule-based refinement of petri nets: a survey. In: Ehrig, H., Reisig, W., Rozenberg, G., Weber, H. (eds.) Petri Net Technology for Communication-Based Systems: Advances in Petri Nets. Lecture Notes in Computer Science, vol. 2472, pp. 161–196. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-40022-6_9
44. Winskel, G.: Petri nets, algebras, morphisms, and compositionality. Inform. Comput. **72**(3), 197–238 (1987). https://doi.org/10.1016/0890-5401(87)90032-0
45. Decker, G., Barros, A.: Interaction modeling using BPMN. In: ter Hofstede, A., Benatallah, B., Paik, H.Y. (eds.) Business Process Management Workshops. Lecture Notes in Computer Science, vol. 4928, pp. 208–219. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78238-4_22
46. Campagna, D., Kavka, C., Onesti, L.: BPMN 2.0 and the service interaction patterns: can we support them all? In: Software Technologies. Communications in Computer and Information Science, vol. 555, pp. 3–20. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25579-8_1
47. Decker, G., Puhlmann, F., Weske, M.: Formalizing service interactions. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) Business Process Management. Lecture Notes in Computer Science, vol. 4102, pp. 414–419. Springer, Heidelberg (2006). https://doi.org/10.1007/11841760_32
48. van der Aalst, W., Mooij, A., Stahl, C., Wolf, K.: Service interaction: patterns, formalization, and analysis. In: Bernardo, M., Padovani, L., Zavattaro, G. (eds.) SFM 2009: Formal Methods for Web Services. Lecture Notes in Computer Science, vol. 5569, pp. 42–88. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01918-0_2
49. Bernardinello, L., Lomazova, I., Nesterov, R., Pomello, L.: Compositional discovery of workflow nets from event logs using morphisms. In: Proceedings of the International Workshop on Algorithms and Theories for the Analysis of Event Data 2018, CEUR Workshop Proceedings, vol. 2115, pp. 23–38. CEUR-WS.org (2018)
50. Nesterov, R., Lomazova, I.: Asynchronous interaction patterns for mining multi-agent system models from event logs. In: Proceedings of the MACSPro Workshop 2019, CEUR Workshop Proceedings, vol. 2478, pp. 62–73. CEUR-WS.org (2019)

Roman Nesterov is a lecturer at the Faculty of Computer Science, HSE University, Russia. In 2021, he finished joint postgraduate studies at the HSE University and the University of Milano-Bicocca, Italy. In 2017, he received the M.Sc. degree in software engineering from HSE University. His research interests are the theory of concurrency, Petri nets, process mining, and formal methods for modeling multi-agent information systems.

Luca Bernardinello graduated in Computer Science at the University of Milano in 1986. Since then, he has worked at the University of Milano, at IRISA-Rennes, at the Joint Research Center of the European Commission. Since 2001, he is a researcher at the Department of Informatics, Systems and Communications (DISCo), University of Milano-Bicocca, Italy. His research activity focuses on the theory of concurrency, and of formal models of concurrent systems. In the field of Petri nets, he has been working on the theory of regions and on the synthesis problem.

Irina Lomazova is a professor at the Faculty of Computer Science, HSE University, Russia. She received the Ph.D. degree in mathematics from Sobolev Institute of Mathematics, Russia in 1982 and the Doctor of Science degree in theoretical informatics from Dorodnitsyn Computing Centre, Russia in 2002. Her research interests include formal methods for modeling and analysis of distributed multi-agent systems and process mining.

Lucia Pomello is an associate professor at the Department of Informatics, Systems and Communications (DISCo), University of Milano-Bicocca, Italy. Graduated in mathematics, she received the Ph.D. degree in computer science from the Universities of Milano and Torino in 1988. Her research interests mainly include formal methods for the analysis and design of distributed systems, the theory of concurrency and of Petri nets.