Discovering Process Models from Event Logs of Multi-Agent Systems Using Event Relations

Anastasiya A. Sherstyugina, Roman A. Nesterov
HSE University
11 Pokrovsky Bulvar, Moscow, Russia
Email: aasherstyugina@edu.hse.ru, rnesterov@hse.ru

***Abstract*—The structure of a process model directly discovered from an event log of a multi-agent system often does not reflect the behavior of individual agents and their interactions. We suggest analyzing the relations between events in an event log to localize actions executed by different agents and involved in their asynchronous interaction. Then, a process model of a multi-agent system is composed from individual agent models between which we add channels to model the asynchronous message exchange. We consider agent interaction within the acyclic and cyclic behavior of different agents. We develop an algorithm that supports the analysis of event relations between different interacting agents and study its correctness. Experimental results demonstrate the overall improvement in the quality of process models discovered by the proposed approach in comparison to monolithic models discovered directly from event logs of multi-agent systems.**

***Index Terms*—Multi-agent systems, event logs, process discovery, Petri nets, event relations, asynchronous interaction.**

I. Introduction

The behavior of an information system is frequently recorded in *event logs*. They can register, for instance, user activities, transaction executions, or message exchanges. An event log consists of finite sequences (traces) of events ordered by the occurrence time. *Process mining* uses event logs to *discover* models reflecting the actual state of processes in an information system. Process models discovered from event logs capture considerable changes that can be introduced to an information system during its operation, while models manually created at the initial life-cycle stages do not take these changes into account [1].

A record in a trace of an event log usually includes not only the identifier of an action, but also other attributes, which can specify the resources necessary for executing the recorded action. These attributes can also designate who executes an action. For example, Table I shows a trace of an event log, where an action record has the "Agent" attribute, and actions are executed by two agents: *Peter* or *Alex*. We say that an event log where actions are attributed with the information on agents records the behavior of a *multi-agent system*.

Process models can be discovered in a variety of notations, including different classes of Petri nets, transition systems, and BPMN (Business Process Model and Notation). In our paper, we focus on modeling the *control-flow* of processes, i.e., the causal dependencies among events in a log. Thus, we will apply *Petri nets* [2] — the formalism extensively used to model and analyze the properties of process behavior.

TABLE I
A trace in an event log of a multi-agent system

Timestamp	Action	Agent
30-12-2022:14.45	prepare msg	Peter
05-01-2023:09.34	send msg	Peter
07-01-2023:12.12	receive msg	Alex
12-01-2023:13.25	send ack	Alex
12-01-2023:14.55	receive ack	Pete
12-01-2023:14.55	local check	Alex

Petri nets are also a convenient tool to model the interaction between different components in a multi-agent system. Figure 1 shows two Petri nets N_1 and N_2 representing two agents with the sequential behavior. They exchange messages through two distinguished channel nodes a and b.

Recent papers in the field of process mining also demonstrate the shift in a focus to a discovery of process models with an understandable structure reflecting the complex synchronizations between objects [3], the hierarchy of activities [4, 5], or the interaction-oriented viewpoints of the architecture of a multi-agent system [6].

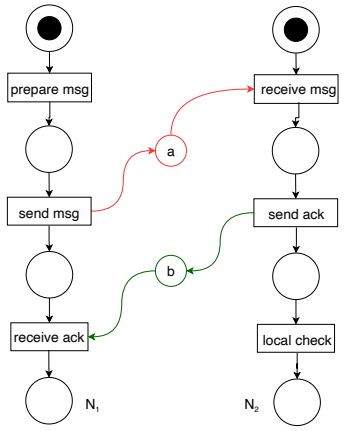

Fig. 1. A multi-agent system with two asynchronously interacting agents

The paper [6] proposed a *compositional* approach to discovering an *architecture-aware* process model from an event log of a multi-agent system. The structure of an architecture-aware process model explicitly reflects agent behavior and their interactions similar to Fig. 1, where two agents exchange

message through channels a and b. A model is constructed by a composition of individual agent models controlled by a manually selected *interface pattern* model. An interface pattern provides a high-level specification of agent interactions. However, in the case of the poor selection of an interface model, one has to reconfigure it and perform an additional check of a reconfigured model.

Here, we propose to ease this restriction on making the preliminary choice of an interface pattern. We suggest to identify asynchronous agent interactions using causal relations between events extracted directly from an event log of a multi-agent system. For instance, in an event log obtained by simulating a process model shown in Fig. 1 the occurrence of "send msg" action will always be recorded before the occurrence of "receive msg" action. Extracting such causality relations will help us to localize events in a log corresponding to the occurrence of actions executed by different agents and involved in their asynchronous communication. Correspondingly, we will determine transitions in individual agent models to be connected via an asynchronous channel.

Note that the automated discovery of process models from event logs is supported by a wide range of algorithms [7]. They usually deal with typical problem of event data representation, including, for instance, *noise* (missing or duplicated records) and *incompleteness*, i.e., a finite event cannot cover all possible process executions. The paper [6] also stressed that an event log of a multi-agent system requires the additional inspection of agent behavior, since the direct discovery from a multi-agent system event log produces process models the structure of which does not explicitly reflect agent behavior as sub-models and agent interactions as distinguished nodes. This happens because the concurrent execution of relatively independent agents leads to a wide range of possible traces recorded in an event log of a multi-agent system.

The quality of discovered process models is the main subject in *conformance checking* [8], which proposes a collection of different dimensions to evaluate the correspondence between an event log and a process model. Fitness and precision are two widely-used quality metrics that can characterize a discovered process model. *Fitness* is an estimation of the ratio of the traces executable by the model to the total number of traces in an event log. A model with the perfect fitness can execute every trace in an event log. For example, the model shown in Fig. 1 can execute the trace in Table I, if we consider N_1 as the behavior of Peter, and N_2 as the behavior of Alex. *Precision* evaluates the ratio of the behavior recorded in an event log and the behavior allowed by a process model. A process model with the perfect precision can only execute traces in an initial event log. The perfect precision limits the use of a discovered process model, since any event log of an information system represents only a finite "snapshot" of all possible process executions.

An architecture-aware process model discovered from an event log of a multi-agent system using the compositional approach of [6] is guaranteed to possess the perfect fitness. The approach to the analysis of agent interactions using causal event relations in a log, proposed in our study, will also ensure the perfect fitness of the process model of a multi-agent system obtained by connecting individual agent models via asynchronous channels.

The main results presented in this paper are:

1) An approach to the analysis of causality relations between events in an event log of a multi-agent system for the identification of specific events involved in the asynchronous communication between different agents.
2) Demonstration of the approach correctness and its experimental evaluation.

The remainder of this paper is organized as follows. In the next section, we collect the formal background of our approach to the analysis of event relations in an event log, including generalized workflow nets (GWF-nets) — a class of Petri nets used to model the behavior of agents and multi-agent systems. Section III considers the localization of events in an event log corresponding the asynchronous agent interactions within the acyclic agent behavior. Section IV explores the case of localizing asynchronous interactions among agents with cycles. Section V reports the outcomes from the experimental evaluation. In Section VI, we review the related research, and Section VII concludes the paper.

II. Background

In this section, we aim to provide the basic definitions concerning several general notions, event logs, and generalized workflow nets. We refer to these definitions when describing our approach to the analysis of causal event relations involving different agents.

S^+ denotes the set of all finite non-empty sequences over a finite 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 $\sigma|_{S'} = bbccb$. If $s \in S$ occurs in a sequence $\sigma \in S^*$, then we write $s \in \sigma$.

$\mathbb{N}$ 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 $s \in S$.

A. Event Logs

An event log is the main input to a process discovery algorithm. It contains a multiset of *traces* — ordered event sequences.

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

When we consider an event log of a multi-agent system with two asynchronously interacting agents, the set $\mathcal{A}$ can be

partitioned into two disjoint subsets, i.e., $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$, s.t. $\mathcal{A}_1 \cap \mathcal{A}_2 = \varnothing$, where $\mathcal{A}_1$ ($\mathcal{A}_2$) is the set of actions executed only be the first (second) agent.

To discover an individual agent model from an event log L of a multi-agent system, we need two project all traces in L onto the set of actions executed by the corresponding agent. The projection of an event log L over $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$ on $\mathcal{A}_1$ is denoted by $L_{\mathcal{A}_1}$. Constructing $L_{\mathcal{A}_1}$ requires projecting every trace $\sigma \in L$ on $L_{\mathcal{A}_1}$, i.e., taking $\sigma|_{\mathcal{A}_1}$. We take into account only non-empty projections $\sigma|_{\mathcal{A}_1}$ and pay additional attention to coinciding projections.

For example, a trace shown in Table I can be projected onto the set of actions executed only by Peter or by Alex.

Let us consider basic causality relations between events recorded in a log L over $\mathcal{A}$, which are determined by the order of corresponding records in the traces of L. Thus, two events $a_1, a_2 \in \mathcal{A}$ are:

1) in the *precedence* relation (a_1 precedes a_2), denoted $a_1 < a_2$, iff $\forall \sigma \in L$: if $a_1, a_2 \in \sigma$, then $\sigma = \sigma' a_1 \sigma'' a_2 \sigma'''$, where $\sigma', \sigma'', \sigma''' \in (\mathcal{A} \setminus \{a_1, a_2\})^*$;
2) in the *following* relation (a_1 follows a_2), denoted $a_1 > a_2$, iff $\forall \sigma \in L$: if $a_1, a_2 \in \sigma$, then $\sigma = \sigma' a_2 \sigma'' a_1 \sigma'''$, where $\sigma', \sigma'', \sigma''' \in (\mathcal{A} \setminus \{a_1, a_2\})^*$;
3) in the *parallel* relation (a_1 is in parallel with a_2), denoted $a_1 >< a_2$, iff there exists a trace $\sigma \in L$, s.t. $\sigma = \sigma' a_1 \sigma'' a_2 \sigma'''$, and a trace $w \in L$, s.t. $w = w' a_2 w'' a_1 w'''$, where $\sigma', \sigma'', \sigma''', w', w'', w''' \in (\mathcal{A} \setminus \{a_1, a_2\})^*$.

It follows that the precedence and the following relations are transitive. For example, $a_1 < a_2$ and $a_2 < a_3$ together leads to traces of the form $\sigma = ...a_1...a_2...a_3...$, which implies $a_1 < a_3$. If required by the context, we can also use the $<_L$ relation sign to explicitly show to which event log this relation corresponds.

B. Generalized Workflow Nets

Workflow nets (WF-nets) [9] are among basic process models discovered from event logs. A WF-net is a special class of a Petri net with the distinguished initial and final places. The execution of a trace in an event log directly corresponds to the execution of a WF-net from its initial to its final place.

We will use generalized workflow nets (GWF-nets), as in [6], to model the behavior of agents and multi-agent systems. Here, we define GWF-nets and their behavior.

Definition 2 (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 our study, we consider nets without self-loops, i.e., $\forall x \in P \cup T$: ${}^\bullet x \cap x^\bullet = \varnothing$ and isolated transitions, i.e., $\forall t \in T$: $|{}^\bullet t| \geq 1$ 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., $N(Y) = (P \cap Y, T \cap Y, F \cap (Y \times Y))$.

Let $N = (P, T, F)$ be a net, and $t_1, t_2 \in T$. Transitions t_1 and t_2 are in *conflict* iff ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \varnothing$. N is *conflict-free* iff no transitions are in conflict.

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 3 (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 [9] 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 (no arcs incoming to corresponding places) and distinguish its final marking (places without outgoing arcs). Compared to a classical WF-net, initial and final marking in a GWF-net can be sets of places rather than singletons.

Definition 4 (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) $\forall x \in P \cup T \, \exists s \in m_0 \, \exists f \in m_f \colon (s, x), (x, f) \in F^{RT}$ *where F^{RT} is the reflexive transitive closure of F.*

According to the third requirement in Definition 4, any node in a GWF-net lies on a path from a place in its initial marking to a place in its final marking. For instance, the Petri net shown earlier in Fig. 1 is a GWF-net, while the behavior of agents N_1 and N_2 can be considered as classical WF-nets with the single initial and final places.

III. Localizing Acyclic Agent Interactions

Here we discuss our approach to finding pairs of actions in an event log representing sending and receiving operations executed by *different* agents. Given an event log of a multi-agents system, we construct a matrix representation of event relations. Then we show how to identify the candidate pairs of events that may represent the asynchronous communication of different agents and connect corresponding transitions in the individual agent models.

A. Matrix Representation of Event Relations

Matrix representation of relations among events recorded in an event log facilitate the pair-wise analysis of events. For what follows, we consider the basic case of a multi-agent system with the *sequential* agent behavior, s.t., actions executed by a specific agent are recorded in an event log only in the precedence or in the following relation. We also show how our reasoning can be extended to agents with parallel and alternative behavioral constructs.

Let L be an event log over $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$, s.t. $\mathcal{A}_1 \cap \mathcal{A}_2 = \varnothing$. Correspondingly, $\mathcal{A}_1$ and $\mathcal{A}_2$ are two disjoint sets of actions executed by two asynchronously interacting agents. Assume $|\mathcal{A}_1| = m$ and $|\mathcal{A}_2| = n$.

We construct matrix R^L of size $m \times n$, which stores relations between the pairs of events representing the occurrence of actions executed by different agents. Given $a_i^1 \in \mathcal{A}_1$ and $a_i^2 \in \mathcal{A}_2$ with $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$, every element r_{ij} in R^L is defined by the following cases:

1) $r_{i,j} =$ "$<$" iff $a_i^1 <_L a_j^2$;
2) $r_{i,j} =$ "$>$" iff $a_i^1 >_L a_j^2$;
3) $r_{i,j} =$ "$><$" iff $a_i^1 ><_L a_j^2$.

Thus, event relations extracted from an event log L fully determines the values of the elements in the corresponding matrix R^L.

Figure 2 shows the example of a matrix representation for event relations constructed from an event log a multi-agent system with asynchronously interacting agents, where the first agents executes actions from the set $\mathcal{A}_1 = \{a_0, a_1, a_2\}$, and the second agent executes actions from the set $\mathcal{A}_2 = \{b_0, b_1, b_2, b_3\}$. For the convenience of the representation, we use names of actions instead of the indices of rows and columns. This matrix says that, for example, in all traces of the initial event log L, actions b_1 and a_2 are executed concurrently (independently), while action a_1 always precedes action b_1.

	b_0	b_1	b_2	b_3
a_0	$><$	$<$	$<$	$<$
a_1	$><$	$<$	$<$	$<$
a_2	$>$	$><$	$><$	$><$

Fig. 2. A matrix of event relations between two asynchronously interacting agents

In addition, recall that agent behavior is considered to be conflict-free and sequential. Then we can easily order actions executed by the same agent according to the event relations, i.e., using the precedence relation. For instance, in Fig. 2, we have that $a_0 < a_1 < a_2$ and $b_0 < b_1 < b_2 < b_3$. This ordering of actions is done before constructing a matrix of event relations. It will help us simplify the further processing and identification of events representing the occurrence of sending-receiving operations between two agents.

The intuition behind the asynchronous message exchange is rather straightforward. After putting a message to a channel, an agent can freely continue its job, while the other agent expecting to receive a message cannot continue to operate until the message is delivered.

This reasoning can also be shifted to our matrix representation of event relations. In a matrix of event relations constructed out of an event log of a multi-agent system with two sequential asynchronously interacting agents, we will be able to locate a "rectangle" formed by the adjacent rows and columns filled by the same event relation "$<$" or "$>$". This is justified by the fact that in all traces of an initial event log several events corresponding to the actions executed by the agent receiving a message are recorded strictly after several events corresponding to the actions executed by the agent who sends a message. Rectangular sections in an event relation matrix filled by the same precedence or following relation are called *regions*.

Definition 5. *Let L be an event log over $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$, s.t. $\mathcal{A}_1 \cap \mathcal{A}_2 = \varnothing$, $|\mathcal{A}_1| = m$, $|\mathcal{A}_2| = n$. Let R^L be an event relation matrix constructed as described above. A rectangular section in R^L formed by k adjacent rows $i, i+1, ..., i+k-1$ and by ℓ adjacent columns $j, j+1, ..., j+\ell-1$ is a p-region (f-region) of R^L if and only if for all $i' = i, i+1, ..., i+k-1$ and $j' = j, j+1, ..., j+\ell-1$ we have that $r_{i',j'}$ ="$<$" ($r_{i',j'}$ ="$>$").*

The region in an event relation matrix R^L starting from row a, column c and finishing at row b and at column d is briefly denoted by $R^L(a-b, c-d)$.

Note that we do not consider a region which is included in another one. We are looking for *maximal* regions in an event relation matrix. For instance, in the event relation matrix shown in Fig. 2, region $R^L(a_2-a_2, b_0-b_0)$ is maximal, since it cannot be extended with other adjacent rows and columns, while $R^L(a_0-a_1, b_1-b_2)$ is not maximal, since it is a part of the bigger region $R^L(a_0-a_1, b_1-b_3)$ that is indeed maximal.

Further, while analyzing regions in an event relation matrix, we always consider maximal regions that cannot be extended with more adjacent rows and columns.

Let us take a closer look at the p-region $R^L(a_0-a_1, b_1-b_3)$ in the event relation matrix shown in Fig. 2. The occurrences of actions a_0 and a_1 were recorded before the occurrences of

actions b_1, b_2 and b_3 in an event log L. Taking into account the sequential agent behavior, i.e., $a_0 < a_1 < a_2$ and $b_0 < b_1 < b_2 < b_3$, we can easily simplify three event relations $a_0 < b_1$, $a_0 < b_2$ and $a_0 < b_3$ to the single relation $a_0 < b_1$, which automatically ensures the remaining two relations. By analogy, three relations $a_1 < b_1$, $a_1 < b_2$ and $a_1 < b_3$ are simplified to $a_1 < b_1$. Finally, two relations $a_0 < b_1$ and $a_1 < b_1$ with $a_0 < a_1$ give us the single event relation $a_1 < b_1$.

Thus, the p-region $R^L(a_0 - a_1, b_1 - b_3)$ in the event relation matrix from Fig. 2 can be fully described by the single event relation $a_1 < b_1$ — the lower left corner of the corresponding rectangular area in the event relation matrix.

Event relation that fully describes a region in an event relation matrix is called the *minimum* of a region, i.e., other event relations within this region coincides with the minimum. It is easy to see that, if the minimum of a p-region is its lower left corner, then the minimum of an f-region is its upper right corner, as illustrated in Fig. 3, where the minimum is highlighted in red.

The minimum event relation in a region is the pair of events which can represent the occurrence of actions agents use for the asynchronous communication.

	...	b_j	...	$b_{j+\ell-1}$	...
...					
a_i		$<$	...	$<$	
...		$<$	...	$<$	
a_{i+k-1}		$<$	...	$<$	
...					

	...	b_j	...	$b_{j+\ell-1}$	...
...					
a_i		$>$	...	$>$	
...		$>$	...	$>$	
a_{i+k-1}		$>$	...	$>$	
...					

Fig. 3. Localizing minimum in a region of an event relation matrix

For example, the event relation matrix R^L shown in Fig. 2 has the p-region $R^L(a_0 - a_1, b_1 - b_3)$ with the minimum relation $a_1 < b_1$ and the f-region $R^L(a_2 - a_2, b_0 - b_0)$ with the minimum relation $a_2 > b_0$. The sequential behavior of corresponding agents can be easily represented via a Petri net with consequent transitions (see N_1 and N_2 in Fig. 4).

According to the minimal event relation of region in the event relation matrix R^L from Fig.2, we introduce two channel places between transitions a_1, b_1 (green place) and transitions b_0, a_2 (red place). Arcs connecting these places with transitions in Fig. 4 follow the direction of the corresponding minimum event relation.

In the following paragraph, we propose an algorithm, which identifies regions in the event relation matrix and finds their corresponding minimal event relations. We prove the algorithm correctness from the point of view of preserving the perfect fitness. We also show that there can be redundant minimum event relations representing different overlapping regions.

B. Algorithm for Finding Minimal Event Relations in Regions of an Event Relation Matrix

We start with an event log L over $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$ of a multi-agent system with two asynchronously interacting agents. Let $\mathcal{A}_1 = m$ and $\mathcal{A}_2 = n$. To simplify the processing of traces in L, we will construct a square event relation matrix R_0^L of size $(m+n) \times (m+n)$ storing event relations between *all* possible pairs of events in $\mathcal{A}$. The indices of an element r_{ij}^0 in R_0^L will directly correspond the indices of actions a_i and a_j in $\mathcal{A}$. Afterwards, choosing necessary rows and columns in a square R_0^L representing the behavior of different agents, we will be able to easily form a required event relation matrix R^L, as described in the previous paragraph.

Here, instead of directly using relation signs, we will assign numbers: -1 for $<$ (precedence), 1 for $>$ (following), and 0 for $><$ (parallel). Initially, R_0^L is filled by the ordering of indices, where $i, j = 1, 2, ..., m+n$: (a) if $i < j$, then $r_{ij} = -1$; (b) $i > j$, then $r_{ij} = 1$. We do not care about the values in R_0^L at its main diagonal (for $r_{i,i}^0$), since we do not consider the reflexive event relations.

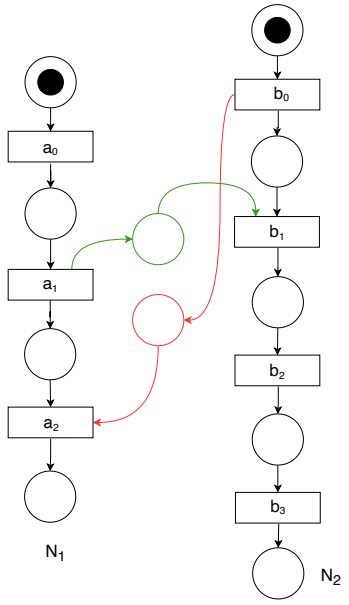

Fig. 4. Introducing channel places according to the matrix from Fig. 2

Subsequently, we update R_0^L according to the actual relations between event pairs in L. Algorithm 1 shows how we process traces in L to extract corresponding event relations.

Given a trace σ in an event log L, we consider every pair of two events preceding each other in σ and update r_{ij}^0 to 0 only if it was 1 before, taking into account that actions executed by different agents are also sorted by the preceding relation. This intuitively means that we have the pair of events recorded in both following and precedence relation in a log representing the sequentialization of parallel execution.

Algorithm 1: Populating an event relation matrix

Input: L – an event log over $\mathcal{A} = \{a_1, a_2, ..., a_{m+n}\}$, R_0^L – an initial square even relation matrix
Output: R_0^L, where $r_{i,j}^0 = -1$ if $a_i <_L a_j$; $r_{i,j}^0 = 0$ if $a_i >\!<_L a_j$; $r_{i,j}^0 = 1$ if $a_i >_L a_j$

```
foreach σ ∈ L do
    foreach a_i, a_j ∈ A, s.t. σ = σ'a_iσ''a_jσ''' do
        if r^0_{i,j} = −1 or r^0_{i,j} = 0 then
            continue
        end
        if r^0_{i,j} = 1 then
            r^0_{i,j} = 0
        end
    end
end
```

For instance, Fig. 5 shows the square event relation matrix R_0^L, built according to Algorithm 1, corresponding to the earlier discussed R^L (see Fig. 2). The main diagonal in this R_0^L is filled with asterisk signs, since we ignore reflexive relations.

We filled two areas in this square matrix with different colors to demonstrate two possible ways of choosing rows and columns for further analysis of event relations corresponding to the occurrence of actions executed by different agents. It is also easy to refine the notion of a region w.r.t. the numerical representation of event relations.

	a_0	a_1	a_2	b_0	b_1	b_2	b_3
a_0	$*$	-1	-1	0	-1	-1	-1
a_1	1	$*$	-1	0	-1	-1	-1
a_2	1	1	$*$	1	0	0	0
b_0	0	0	-1	$*$	-1	-1	-1
b_1	1	1	0	1	$*$	-1	-1
b_2	1	1	0	1	1	$*$	-1
b_3	1	1	0	1	1	1	$*$

Fig. 5. A square matrix of event relations constructed by Algorithm 1

The p-region is the rectangular area of the numerical event relation matrix filled completely with -1, while the f-region should be filled only with 1. Here, we also consider maximal region only, which fully correspond to the representation discussed in the previous paragraph.

Let us consider another example of an event relation matrix R^L, shown in Fig. 6, constructed from an event log L over $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$, where $\mathcal{A}_1 = \{x_0, x_1, x_2, x_3\}$ and $\mathcal{A}_2 = \{y_0, y_1, y_2\}$.

In this event matrix, there are two p-regions $R^L(x_0 - x_1, y_0 - y_2)$ with the minimum event relation $x_1 < y_0$ and $R^L(x_0 - x_3, y_0 - y_1)$ with the minimum event relation $x_3 < y_0$. However, since $x_0 < x_1 < x_2 < x_3$, there is enough to keep $x_3 < y_0$, which will automatically satisfy $x_1 < y_0$ because x_3 occurs after x_1. This agrees with the transitivity of the precedence relation. The redundancy of these event relations can be easily shown in the corresponding agent models (see Fig. 7). We do not need to add a place between transitions x_1 and y_0 having a place between transitions x_2 and y_0.

Transition x_3 will fire only after transition x_1. Thus, adding the direct channel place between transitions x_1 and y_1 will not introduce new event relations different from those already present in the matrix from Fig. 6, unless this channel is not necessary according to the practical requirements.

	y_0	y_1	y_2
x_0	-1	-1	-1
x_1	-1	-1	-1
x_2	-1	-1	0
x_3	-1	-1	0

Fig. 6. An event relation matrix with two overlapping p-regions

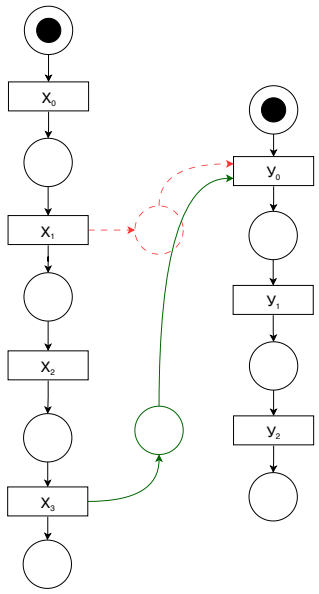

Fig. 7. Redundant channel according to the event matrix shown in Fig. 6

The same transitivity principle can also be applied to the case of two overlapping f-regions. The example of an event relation matrix with two overlapping f-regions is shown in Fig. 8. The minimum event relation $x_0 > y_3$ will cover all event relations in both f-regions.

	y_0	y_1	y_2	y_3
x_0	0	0	1	1
x_1	0	0	1	1
x_2	1	1	1	1
x_3	1	1	1	1

Fig. 8. An event relation matrix with overlapping f-regions

Note that the localization of the minimum in a region of an event relation matrix R^L actually boils down to finding the cell $r_{i,j}$, s.t.:

- if $r_{i,j} = -1$, where $r_{i+1,j} \neq -1$ and $r_{i,j-1} \neq -1$, then $r_{i,j}$ is the minimum of a p-region in R^L with the corresponding event relation $a_i < a_j$;
- if $r_{i,j} = 1$, where $r_{i-1,j} \neq 1$ and $r_{i,j+1} \neq 1$, then $r_{i,j}$ is the minimum of an f-region in R^L with the corresponding event relation $a_i > a_j$;

Thus, the main scheme for the compositional discovery of a process model from an event log L over $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$ of a multi-agent system using minimal event relations in the event relation matrix R^L includes the following steps:

1) population of the square event relation matrix R^L_0 (Algorithm 1) and selection of columns and rows (for R^L) with the actions corresponding to different agents;
2) identification of minimum event relations in p-regions and f-regions in R^L;
3) discovery of individual agent process models N_1 and N_2 from projected event logs $L_{\mathcal{A}_1}$ and $L_{\mathcal{A}_2}$, respectively;
4) introduction of channel places between transitions in N_1 and N_2 corresponding to the events associated by the minimal event relations constructed at step 2.

Individual agent models can be discovered from projected event logs using any existing process discovery algorithm. We recommend to use *Inductive miner* [10], since it can guarantee the perfect fitness of a discovered model.

The soundness of the compositional discovery procedure presented above is formalized in the following Theorem 1, where we prove that a process model of a multi-agent system inherits the perfect fitness of agent models discovered from projected event logs. In other words, a process model obtained by adding channel places between transitions in the individual agent models with respect to the minimal event relations can execute all traces in the event log L of a multi-agent system.

Theorem 1. *Let L be an event log of a multi-agent system over $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$. Let $E \subseteq (\mathcal{A}_1 \times \mathcal{A}_2) \cup (\mathcal{A}_2 \times \mathcal{A}_1)$ be the set of event pairs, which correspond to the minimum event relations extracted from the event relation matrix R^L. If N_i is a GWF-net discovered from the projection $L_{\mathcal{A}_i}$, such that it perfectly fits $L_{\mathcal{A}_i}$ with $i = 1, 2$, then N obtained from N_1 and N_2 by introducing channel places between transition pairs corresponding to event pairs in E perfectly fits L as well.*

Proof. The proof is done by contradiction. Assume $N = (P, T, F, m_0, m_f)$ does not perfectly fits L. Consider a pair $(a_i, a_j) \in E$, which corresponds to the minimal event relation $a_i < a_j$. Let $\sigma \in L$ be a trace of the event log L, which contains a_i and a_j, which N cannot execute. Since $a_i < a_j$, $\sigma = \sigma' a_i \sigma'' a_j \sigma'''$. Transitions $t_i, t_j \in T$ corresponding to events a_i and a_j are connected in N, such that there is a place $c \in P$, where $(t_i, c), (c, t_j) \in F$. If N cannot execute σ, then transition t_j should be able to fire before t_i, which will result in $\sigma = \sigma' a_j \sigma'' a_i \sigma'''$. This contradicts the correct configuration of the trace $\sigma = \sigma' a_i \sigma'' a_j \sigma'''$. Thus, the initial assumption that N does not perfectly L is wrong. Hence, N obtained by adding corresponding channels between transitions in N_1 and N_2 perfectly fits L. □

Here, we considered the analysis of acyclic interactions between agents with sequential and conflict-free behavior. However, we can also generalize our approach to agents with conflicting (alternative) and parallel branches.

It is necessary to extend the proposed collection of event relations with the *conflicting* relation. Two actions a_1 and a_2 are in conflict (denoted by $a_1 \# a_2$ and 2 for the square matrix R^L_0) if for every trace in an event log a_1 and a_2 never occur together. Conflicting and parallel actions can be involved in the asynchronous interaction among agents.

Application of our approach requires separate investigation of sequential parts in agent behavior recorded in a log for the proper construction of regions in the corresponding matrix with ordered actions. This is by analogy with the identification of sequential components in GWF-nets (recall the state machine decomposability discussed in Section II).

For example, Fig. 9 shows the acyclic interaction between N_1 and N_2, where N_1 has the conflict between transitions x_3 and x_5. In an event log, actions x_3 and x_5 will never occur in the same trace. Using R^L_0 we can identify maximal sequential parts in the behavior of N_1, i.e., $x_2 < x_3$ and $x_4 < x_5$, and construct two inter-agent matrices to localize minimal event relations in corresponding regions. Two minimal event relations $y_2 < x_3$ and $y_2 < x_5$ with the common event y_2 are ensured with a single channel place a connecting transitions w.r.t. the relation direction.

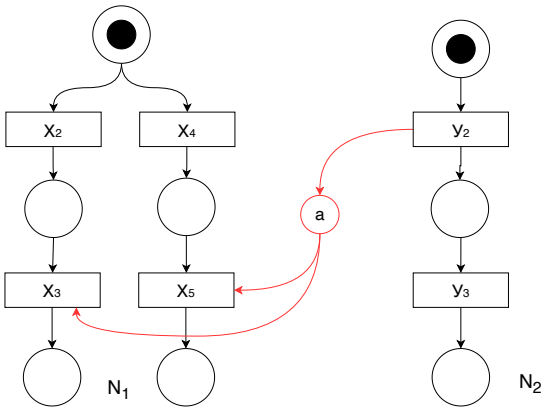

Fig. 9. Acyclic interaction with choice in the agent behavior

Using a similar reasoning, we can analyze asynchronous interactions involving different parallel branches in the behavior of agents. In this case, the minimal relations with the common events are modeled by individual channel places, since, for parallel actions, the occurrence of one does not exclude the occurrence of the others.

In the following section, we consider asynchronous interactions among agents, s.t. actions used for the message exchange are involved in a cycle. The direct analysis of causality relation is not enough for cyclic behavior, since events within a cycle can be recorded in an event log in any order.

IV. Localizing Cyclic Agent Interactions

In this section, we consider the problem of identifying the pairs of events in an event log of a multi-agent system involved intro the cyclic interaction between different agents. Cyclic interaction implies that the actions corresponding to the asynchronous message exchange are executed within a cycle in the agent behavior. We cannot directly use the minimal causality relations proposed in the previous section, since actions within cycles in different agents will be recorded in an event log in any order.

A. Bounded Asynchronous Channels

The cyclic interaction is directly connected with the problem of the boundedness in Petri net theory. Consider an example of cyclic interaction shown in Fig. 10. The cycle in N_1 sends messages to the cycle in N_2 via the single channel a.

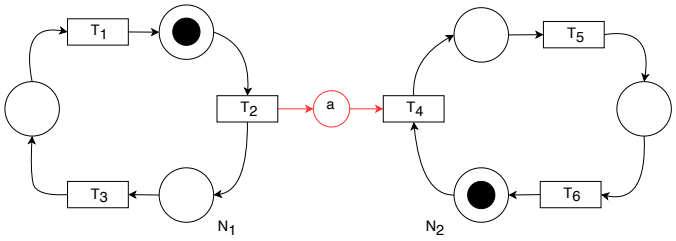

Fig. 10. An unbounded asynchronous channel

The problem with this channel place a is that N_1 can put messages to place a infinitely many times, which will lead to the possibility of the unbounded number of messages in a. As a result, the complete system will have infinitely many different reachable states.

To avoid the problem of the unboundedness, we can introduce an additional place into the model of a multi-agent system with two interacting agents. This place will act as a "limiter" of the number of messages an asynchronous channel can store.

For example, if we add place b, as shown in Fig. 11, the maximum number of messages that can be put to place a by N_1 will not exceed 1. Such places are called *complement* in Petri nets, since they mirror the direction of arcs connected with the channel place.

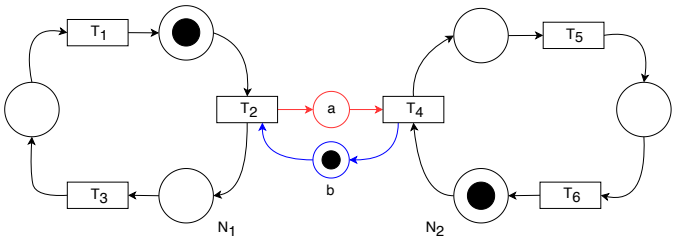

Fig. 11. An asynchronous channel with the bound

In fact, the number of tokens in the complement place we add to bound an asynchronous channel correspond to the maximum number of messages this asynchronous channel can store. In the following paragraph, we show our approach to the analysis of cyclic interactions between agents in a multi-agent system with respect to the maximum number of messages a candidate asynchronous channel place can store.

B. Algorithm for Localizing Cyclic Asynchronous Interactions and Channel Bounds

In the case of the cyclic asynchronous interactions, we cannot directly refer to the minimum event relations, since all involved actions can potentially be recorded in any order in an event log. For example, by simulating the net from Fig. 11, we can obtain $t_2 < t_4$ as well as $t_4 < t_2$. Instead, we are going to consider the number of occurrences of events in an event log to devise the maximum number of messages an asynchronous channel can handle.

For what follows, let L be an event log of a multi-agent system with two asynchronously interacting agents over $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$. We isolate only the cyclic behavior of agents in these sets $\mathcal{A}_1$ and $\mathcal{A}_2$, since the acyclic part can be analyzed before using the algorithm described in Section III. To avoid the ambiguity, we assume additionally that actions $\mathcal{A}_1$ represent the behavior of an agent sending messages, while the actions $\mathcal{A}_2$ — the behavior of an agent receiving messages.

The main idea of our approach is to analyze pairs of actions in $\mathcal{A}_1 \times \mathcal{A}_2$ to count the maximum number of messages. If in a trace of L the occurrence of an event $a_1 \in \mathcal{A}_1$ is recorded, then the bound in the number of messages decreases by 1. If in a trace of L the occurrence of an event $a_2 \in \mathcal{A}_2$ is recorded, then the bound in the number of messages increases by 1.

We assume that an asynchronous channels stores $k \geq 0$ messages initially. Algorithm 2 shows how to analyze the pairs of events in $\mathcal{A}_1 \times \mathcal{A}_2$ according to their behavior with respect to increasing and decreasing k. This algorithm produces the range, i.e., the minimum and maximum number of messages an asynchronous channel between a concrete pair of events can process.

Consider the example of using Algorithm 2 for the event log of a multi-agent system L (see Table II) over $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$, where $\mathcal{A}_1 = \{t_4, t_5, t_6\}$ and $\mathcal{A}_2 = \{t_1, t_2, t_3\}$.

TABLE II
An Event Log of a Multi-Agent System with Four Traces

Trace 1	$t_4t_5t_6t_4t_5t_2t_3t_6t_4t_5t_6t_4t_5t_2t_3t_6t_1t_2t_3t_1t_2t_3t_4t_1t_5t_6t_2t_3t_1$ $t_2t_3t_1t_2t_3t_1t_4t_2t_3t_5t_1t_6t_4t_5t_6t_4t_5t_6t_4t_2t_5t_3t_6t_1$
Trace 2	$t_2t_3t_1t_2t_3t_1t_2t_3t_4t_1t_5t_2t_6t_3t_4t_1t_5t_6t_4t_2t_5t_3t_1t_6t_2t_4$
Trace 3	$t_2t_3t_1t_2t_3t_1t_2t_3t_1t_4t_1t_5t_6t_4t_1t_5t_6t_4t_1t_5t_6t_4t_1t_5t_6t_4t_1t_5t_6$
Trace 4	$t_4t_1t_5t_6t_4t_1t_5t_6t_2t_3t_1t_2t_3t_1t_2t_3t_1t_2t_3t_1t_2t_3t_1$

The result of computing the minimum and maximum number of messages for different event pairs in Trace 1 in this event log is presented in Table III.

For instance, we consider the pair of events (t_4, t_1) of transitions between which we aim to add a bounded asynchronous channel place. We check the minimum and maximum number

TABLE III
APPLYING ALGORITHM 2 TO TRACE 1 IN THE LOG FROM TABLE II

Event pair	Minimum	Maximum
(t_1, t_4)	$k-3$	$k+2$
(t_1, t_5)	$k-4$	$k+1$
(t_1, t_6)	$k-4$	$k+2$
(t_2, t_4)	$k-2$	$k+3$
(t_2, t_5)	$k-3$	$k+3$
(t_2, t_6)	$k-2$	$k+3$
(t_3, t_4)	$k-3$	$k+2$
(t_3, t_5)	$k-3$	$k+2$
(t_3, t_6)	$k-3$	$k+2$

of messages for all traces in the event log from Table II, as shown in Table IV.

To cover the complete event log from Table II, we need to construct the range for the channel between events t_4 and t_1 uniting the individual ranges for all traces. Thus, according to Table IV, the range of the number of messages that can be handled by the asynchronous channel between transitions t_4 and t_1 is $[k-3; k+3]$. The length of this range is $k+3-(k-3) = 6$. Therefore, the maximum number of messages that can be stored in the channel between t_4 and t_1 is 6.

Algorithm 2: Analyzing cyclic interactions in a trace

Input: $\sigma \in L$ – a trace in an event log over $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$, where $\mathcal{A}_1 \cup \mathcal{A}_2 = \varnothing$
Output: Minimum $min(p)$ and maximum $max(p)$ number of messages for every $p = (a_1, a_2) \in \mathcal{A}_1 \times \mathcal{A}_2$ a channel between a_1 and a_2 may process

```
foreach a1 ∈ A1 do
    foreach a2 ∈ A2 do
        maxK ← k, minK ← k, current ← k
        foreach e_i ∈ σ = e1 e2 ... en do
            if e_i = a1 then
                current ← k − 1
            end
            if e_i = a2 then
                current ← k + 1
            end
            maxK ← MAX(current, maxK)
            minK ← MIN(current, minK)
        end
    end
    min(a1, a2) ← minK, max(a1, a2) ← maxK
end
```

Note also that, since the left border of this range $k-3$, initially the channel place between t_4 and t_1 should have 3 tokens in it, because the number of tokens in places of a Petri net cannot go below 0. This is also caused by the fact that in Trace 2 of the event log from Table II the agent receiving messages operates before the one who sends messages.

We have everything to construct the model of a multi-agent system with two agents exchanging messages through actions t_4 and t_1 within cyclic sequential behavior regarding the event log from Table II. Fig. 12 shows the corresponding process model for this multi-agent system, where N_1 is the agent sending messages with transitions t_4, t_5, t_6, and N_2 — receiving messages with transitions t_1, t_2, t_3.

TABLE IV
THE NUMBER OF MESSAGES IN THE CHANNEL CONNECTING t_4 AND t_1

	Minimum	Maximum
Trace 1	$k-3$	$k+2$
Trace 2	k	$k+2$
Trace 3	$k-2$	$k+3$
Trace 4	$k-2$	$k+3$

We note that the similar analysis can be done for any pair of transitions representing the behavior of sending and receiving agents, s.t. one can add an asynchronous channel between them in different ways, unless there is an additional information on actions provided. For instance, one can choose those transitions with the channel the capacity of which does not exceed 1 (for safe Petri nets). In addition, as in the case of the acyclic interaction, it is possible to analyze the cyclic behavior of agents with parallel and alternative behavioral constructs inside cycles by checking interactions between separate sequential components.

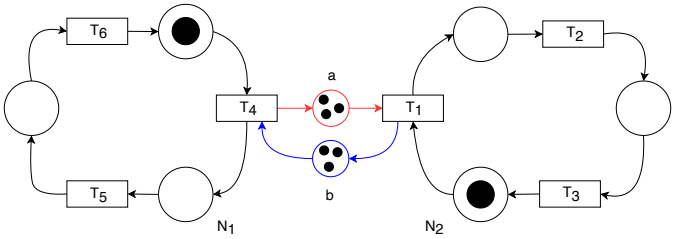

Fig. 12. A multi-agent system with two interacting agents with cyclic behavior

Moreover, the same property on preserving the perfect fitness of the individual agent models (see Theorem 1) will also hold for the cyclic interaction, since we add channel places between transitions in the strict accordance with an initial event log.

V. EXPERIMENTAL EVALUATION

This section reports the key outcomes obtained from the series of experiments conducted to evaluate the proposed approach to the identification of the pairs of events involved into the acyclic and cyclic interactions among different agents in a multi-agent system.

A. Layout of Experiments

We compared process models discovered by our approach and directly from an event log of a multi-agent system. We also considered a specific case of a process model with "disconnected" agents, i.e., we do not add asynchronous channels between them.

Within the experimental evaluation, we used the synthetic event logs of multi-agent systems recording different ways of agent asynchronous interactions provided in [11]. They were

also used to test the compositional approach to discovering architecture-aware process model of multi-agent systems [6]. This dataset was constructed with respect to various widespread service interaction patterns described in [12].

Thus, process models of multi-agent systems obtained by our approach to introducing channels were compared with the following other models:

1) *reference* models, also provided in [11], which represent the ideal model of a multi-agent system with the minimum number of asynchronous channels;
2) *disconnected agent* models, where individual agent models discovered from projected event logs are put together without adding any asynchronous channels;
3) *monolithic* models discovered from directly event logs .

We characterized these models according to the following two quality dimensions:

1) *precision* evaluating the extra amount of behavior allowed by a process models regarding the behavior recorded in an event log (see the gray area in Fig. 13);
2) the number of asynchronous channels connecting transitions in the models of different agents.

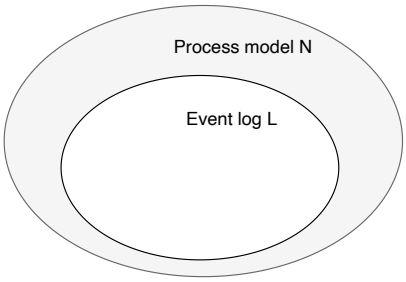

Fig. 13. The behavior of a process model and traces in an event log

The perfect fitness of discovered process models is guaranteed by our approach and by the paper [6]. A model with the disconnected agent behavior also ensures the perfect fitness, since the concurrent execution of fully independent agents can also cover all possible ways of their asynchronous interactions. Therefore, we did not need to measure the fitness of considered process models. As for the precision, we used the approach from [13] as the one, which provides the balanced estimation of this quality dimension. The experimental evaluation was supported by the ProM software [14].

B. Experiment Results and Discussion

Table V reports the results on comparing the quality of process models discovered from an event log of a multi-agent system using our approach with the quality of directly discovered models (monolithic) and models with the disconnected agent behavior. The dataset [11] used in our experiments contains seven different event logs of multi-agent system corresponding to different ways of acyclic (IP-1, ..., IP-6) and cyclic (IP-7) patterns of asynchronous interactions. We also did not evaluate the number of channels in monolithic process models of multi-agent systems, since in the structure of such a model one cannot unambiguously identify the behavior of individual agents and asynchronous channel places.

According to the experimental results provided in Table V, we may conclude the following. Firstly, our approach detects considerably more "points" of the asynchronous interactions between different agents compared to the ideal reference model. A finite sequential record of the concurrent execution of relatively independent agents cannot cover all possible scenarios. Thus, there are more candidate relations among event pairs that can be considered for adding asynchronous channel places between the corresponding transitions. We can further analyze all found minimum event relations from the point of view on their frequencies w.r.t. an initial event log to exclude some of them. Secondly, process models obtained by our approach exhibits the increase in the precision estimations, since introduction of other asynchronous channels decreases the amount of extra behavior allowed by a model and not recorded in a log. Thirdly, we generally outperform the quality of the monolithic process model the structure of do not correspond to the architecture of a multi-agent system regarding the individual agent behavior and their interactions.

We believe that increasing the number of traces in an event log will bring the quality of process models obtained by adding channels using our approach closer to the evaluations of reference models, since an event log will exhibit more different execution scenarios. As one of the possible directions of future research, we will consider the analysis of connections between the precision of agent models and of system models obtained by our approach based on event relation.

VI. Related Work

As we mentioned in Introduction, different algorithms were proposed for the computer-aided discovery of process models from event logs. The most popular ones include Inductive miner [10], Fuzzy miner [15], Region Theory-based miner [16], and Genetic miner [17]. These algorithms can guarantee that discovered process models will exhibit certain properties. For example, Inductive miner can guarantee perfect fitness and soundness of discovered workflow nets. In the recent study [7], the authors gave an extensive review and comparison of process discovery algorithms. Note that these algorithms are aimed to tackle different internal limitations of event data representation rather than to analyze interactions among different information system components.

The quality of discovered process models takes an important part in choosing an algorithm for discovering process models from event logs. Conformance checking [7] provides several dimensions that allow one to evaluate the correspondence between a model and an event log (fitness, precision, generalization), and the structure of a discovered model (simplicity). Researchers stress that there is a lack of universally applicable properties and requirements that can constitute the formal basis for computing conformance checking dimensions [7, 18]. Thus, our study also considers the formal analysis of preserving the perfect fitness of agent models discovered from

TABLE V
EXPERIMENTAL RESULTS: THE NUMBER OF ASYNCHRONOUS CHANNELS AND PRECISION EVALUATION

		Reference model		Disconnected agents	Monolithic model	Our approach	
		Channels	**Precision**	**Precision**	**Precision**	**Channels**	**Precision**
Acyclic interaction	**IP-1**	1	0.7156	0.6949	0.5825	14	0.8109
	IP-2	2	0.4014	0.3719	0.3880	33	0.5337
	IP-3	2	0.7545	0.7097	0.8984	26	0.8861
	IP-4	2	0.7589	0.6752	0.6684	10	0.8420
	IP-5	4	0.3902	0.3503	0.1342	39	0.5724
	IP-6	4	0.5636	0.5256	0.6849	34	0.7034
Cyclic interaction	**IP-7**	3	0.8165	0.5945	0.1327	5	0.6782

filtered logs in a multi-agent system models with introduced asynchronous channels recovered using event relations.

The problem of discovering process models with a clear structure is studied from different perspectives. Inductive miner produces well-structured process models that are recursively constructed from "building blocks" representing standard behavioral constructs: sequential, cyclic, parallel, and alternative execution of actions. A series of papers [19, 20, 21] proposed different approaches to improving the structure of discovered models by the additional localization of the environment of events in a log and by composing fragments of regular and frequent behavior with the rare "exceptional" scenarios. Discovery of hierarchical process models, where a high-level event represents a sub-process, was studied in [4]. The identification of low-level and high-level events in an event log is a natural way to improve the structural representation of a process model. The paper [3] proposed a novel approach to discover object-centric Petri nets from event logs. Interactions of objects is represented through complex synchronizations which allow one to model consumption and production of objects of different types. Compositional discovery of behaviorally correct and "architecture-aware" process models from event logs of multi-agent systems was studied in [6]. Using interface patterns and structural property-preserving mapping helped to achieve the clear structure of a model reflecting independent behavior of agents and their communication.

Our study continues [6] in a way that we are trying to analyze and identify "points" of asynchronous interactions — actions involved in the asynchronous message passing between agents — directly from event logs. Based on the causality relations among events in a log, we can find, for example, pairs of actions that are always executed in a fixed order. Such actions are then considered to be the candidates to represent send-receive operations within the asynchronous interaction. Then we may relax the requirement on the manual selection of interface patterns, as originally proposed.

Patterns are typically used in the software development as the collection of best practices and recurring development scenarios [22]. Frequently used control-flow constructs in business process modeling — workflow patterns — were systematically studied in [23]. In [12, 24], the authors generalized workflow patterns for modeling widespread correct service interactions in complex and large-scale systems. Within the context of process discovery, several papers also proposed different approaches for the analysis of behavioral patterns in event logs, including, among the others, [25, 26], but these patterns were not considered from the point of view of interactions among different information system components.

VII. CONCLUSION

In this paper, we considered the problem of discovering a process model in terms of a generalized workflow net from an event log of a multi-agent system with the understandable structure reflecting the architecture of a system. A model of a multi-agent system is obtained from a composition of individual agent models through the introduction of asynchronous channels. To identify transitions in agent models to be connected via a channel place, we analyze causal relations between events recorded in an event log. Within the asynchronous agent interactions, several actions of one agents are executed before certain actions of the other. This idea helped us to localize the so-called minimum relations between events corresponding to the occurrence of actions executed by different agents. The pairs of events representing these minimum relations can be seen as "points" of the asynchronous communication between agents. Transitions corresponding to these events can be connected with an asynchronous channel place. We also showed that certain minimum event relations can cover other minimum relations between events in a log.

The pair-wise analysis of relations between events recorded in an event log was based on matrices with rows and columns representing events. Matrix representation of event logs was used in process mining in different contexts (cf. the footprint matrix in the basic α-algorithm [27] and the analysis of unchanged sections in BPMN models [28]).

We separately considered the cases of the acyclic and cyclic asynchronous interactions, since, within the latter one, events can be recorded in any possible order. To localize events in the cyclic communication, we analyzed the number of event occurrences regarding the maximum number of messages that a potential asynchronous channel can handle. This allows us to achieve the boundedness, i.e., the finite number of reachable states, in a complete process model of a multi-agent system.

The correctness of the proposed approach to adding asynchronous channels between behavioral models of individual agents is justified by the fact that we preserve the perfect

fitness, i.e., the ability to execute all traces in the event log of a multi-agent system, of agent model in a complete system model. We conducted a series of experiments to evaluate our approach. The experimental results demonstrate the overall improvement in process models discovered by adding asynchronous channels in comparison to models directly discovered from event logs of multi-agent systems.

As for the future research, we plan to continue it in the following directions. Firstly, we would like to consider more complex ways of the asynchronous communications, including, for instance, message broadcasting. Secondly, we also intend to make a deeper analysis of the preservation of behavioral properties, including deadlock-freeness, in a process model of a multi-agent system obtained from individual agent models connected by asynchronous channel places. For example, we need to avoid the introduction of channels leading to the "circular wait", as shown in Fig. 14, where N_1 waits for N_2, while N_2 waits for N_1 at the same time. Finally, we plan to conduct more experiments using real-life event logs.

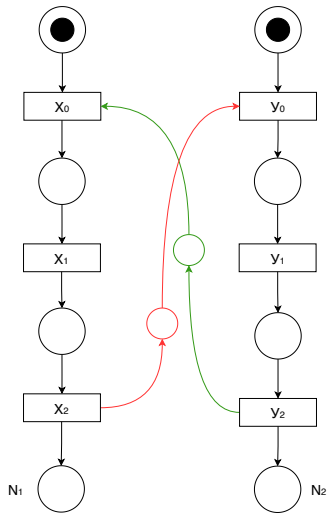

Fig. 14. Asynchronous interaction may result in a deadlock

ACKNOWLEDGMENT

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

REFERENCES

[1] W. van der Aalst, *Process Mining: Data Science in Action*. Springer, Heidelberg, 2016.

[2] W. Reisig, *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer, Heidelberg, 2013.

[3] W. van der Aalst and A. Berti, "Discovering object-centric petri nets," *Fundamenta Informaticae*, vol. 175, pp. 1–40, 2020.

[4] A. Begicheva and I. Lomazova, "Discovering high-level process models from event logs," *Modeling and Analysis of Information Systems*, vol. 24, no. 2, pp. 125–140, 2017.

[5] C. Li, S. van Zelst, and W. van der Aalst, "An activity instance based hierarchical framework for event abstraction," in *2021 3rd International Conference on Process Mining (ICPM)*, 2021, pp. 160–167.

[6] R. Nesterov, L. Bernardinello, I. Lomazova, and L. Pomello, "Discovering architecture-aware and sound process models of multi-agent systems: a compositional approach," *Software & Systems Modeling*, vol. 22, pp. 351–375, 2023.

[7] A. Augusto, R. Conforti, M. Dumas, M. Rosa, F. Maggi, A. Marrella, M. Mecella, and A. Soo, "Automated discovery of process models from event logs: Review and benchmark," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 4, pp. 686–705, 2019.

[8] J. Carmona, B. van Dongen, A. Solti, and M. Weidlich, *Conformance Checking: Relating Processes and Models*. Springer, Cham, 2018.

[9] W. van der Aalst, "Workflow verification: Finding control-flow errors using petri-net-based techniques," in *Business Process Management: Models, Techniques, and Empirical Studies*, ser. Lecture Notes in Computer Science, vol. 1806. Springer, Heidelberg, 2000, pp. 161–183.

[10] S. Leemans, D. Fahland, and W. van der Aalst, "Discovering block-structured process models from event logs – a constructive approach," in *Application and Theory of Petri Nets and Concurrency*, ser. LNCS, vol. 7927. Springer, Heidelberg, 2013, pp. 311–329.

[11] R. Nesterov, "Compositional discovery of architecture-aware and sound process models of multi-agent systems: experimental: data experimental data. (version 1) [data set]." [Online]. Available: https://doi.org/10.5281/zenodo.5830863

[12] A. Barros, M. Dumas, and A. ter Hofstede, "Service interaction patterns," in *Business Process Management*, ser. LNCS, vol. 3649. Springer, Heidelberg, 2005, pp. 302–318.

[13] J. Muñoz-Gama and J. Carmona, "A fresh look at precision in process conformance," in *Business Process Management*, ser. Lecture Notes in Computer Science, vol. 6336. Springer Heidelberg, 2010, pp. 211–226.

[14] "ProM Tools," https://www.promtools.org/doku.php.

[15] C. Günther and W. van der Aalst, "Fuzzy mining – adaptive process simplification based on multi-perspective metrics," in *Business Process Management*, ser. LNCS, vol. 4714. Springer, Heidelberg, 2007, pp. 328–343.

[16] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser, "Process mining based on regions of languages," in *Business Process Management*, ser. LNCS, G. Alonso, P. Dadam, and M. Rosemann, Eds., vol. 4714. Springer, Heidelberg, 2007, pp. 375–383.

[17] W. van der Aalst, A. de Medeiros, and A. Weijters, "Genetic process mining," in *Applications and Theory of Petri Nets*, ser. Lecture Notes in Computer Science, G. Ciardo and P. Darondeau, Eds., vol. 3536. Springer, Heidelberg, 2005, pp. 48–69.

[18] W. van der Aalst, "Relating process models and event logs – 21 conformance propositions," in *Proceedings of the International Workshop ATAED-2018*, ser. CEUR Workshop Proceedings, vol. 2115. CEUR-WS.org, 2018, pp. 56–74.

[19] A. Kalenkova, I. Lomazova, and W. van der Aalst, "Process model discovery: A method based on transition system decomposition," in *Application and Theory of Petri Nets and Concurrency*, ser. LNCS, vol. 8489. Springer, Cham, 2014, pp. 71–90.

[20] A. Kalenkova and I. Lomazova, "Discovery of cancellation regions within process mining techniques," *Fundamenta Informaticae*, vol. 133, pp. 197–209, 2014.

[21] W. van der Aalst, A. Kalenkova, V. Rubin, and E. Verbeek, "Process discovery using localized events," in *Application and Theory of Petri Nets and Concurrency*, ser. Lecture Notes in Computer Science, R. Devillers and A. Valmari, Eds., vol. 9115. Springer, Cham, 2015, pp. 287–308.

[22] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[23] W. van der Aalst, A. ter Hofstede, B. Kiepuszewski, and A. Barros, "Workflow patterns," *Distributed and Parallel Databases*, vol. 14, pp. 5–51, 2003.

[24] D. Campagna, C. Kavka, and L. Onesti, "BPMN 2.0 and the service interaction patterns: Can we support them all?" in *Software Technologies*, ser. Communications in Computer and Information Science, vol. 555. Springer, Cham, 2015, pp. 3–20.

[25] S. Suriadi, R. Andrews, A. ter Hofstede, and M. Wynn, "Event logs imperfection patterns for process mining: Towards a systematic approach to cleaning event logs," *Information Systems*, vol. 34, pp. 132–150, 2017.

[26] M. Acheli, D. Grigori, and M. Weidlich, "Discovering and analyzing contextual behavioral patterns from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 12, pp. 5708–5721, 2022.

[27] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: discovering process models from event logs," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 9, pp. 1128–1142, 2004.

[28] K. Artamonov and I. Lomazova, "What has remained unchanged in your business process model?" in *2019 IEEE 21st Conference on Business Informatics (CBI)*, 2019.