Asynchronous Interaction Patterns for Mining Multi-Agent System Models from Event Logs[⋆]

Roman A. Nesterov[1,2], Irina A. Lomazova[1]

[1] National Research University Higher School of Economics,
20 Myasnitskaya Ulitsa, 101000 Moscow, Russia
[2] Dipartimento di Informatica, Sistemistica e Comunicazione,
Università degli Studi di Milano-Bicocca,
Viale Sarca 336 - Edificio U14, I-20126 Milano, Italia

Abstract. Process models discovered from event logs of multi-agent systems may be complicated and unreadable. To overcome this problem, we suggest using a compositional approach. A system model is composed from agent models w.r.t. an interface. Morphisms guarantee that composition of correct models is correct. This study contributes to the practical implementation of the morphism-based compositional approach. We use interaction patterns to model typical interfaces. Experimental evaluation justifies the practical value of the compositional approach.

Keywords: multi-agent systems, event logs, process discovery, Petri nets, composition, morphisms, interaction patterns

1 Introduction

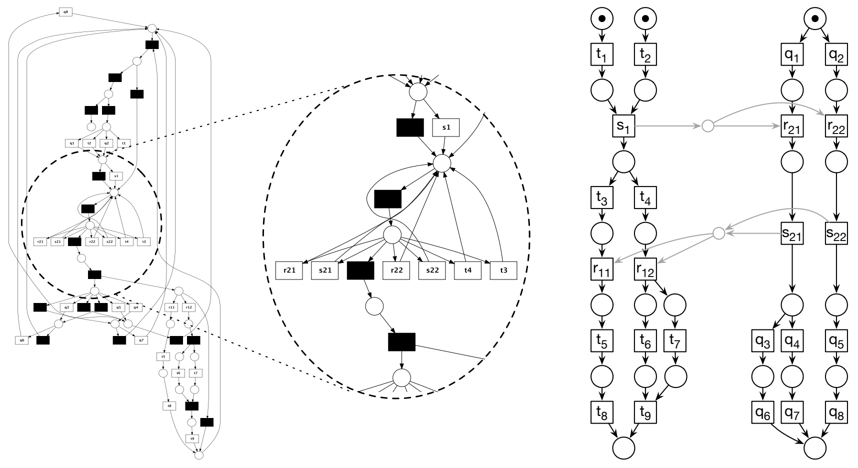
Process discovery provides support for companies in managing and improving their business processes[1]. Nowadays, companies collect significant amounts of event data from different sources. For example, ERP (enterprise resource planning) systems store transaction logs. These *event logs* are used in process discovery to extract *real* process models in contrast to manually created models of idealized processes [2]. A lot of algorithms for automated discovery of process models have been proposed over recent years [3]. They use a plethora of notations to represent process models including Petri nets, heuristic nets or BPMN. In our study, we focus on modeling the *control-flow*, i.e. causal dependencies among process activities. For this purpose we use Petri nets [19] to construct *formal* (executable) process models.

Process models discovered from event logs of *multi-agent systems* (MAS) can be complicated and unreadable. The interaction of several agents produces rather intricate behavior. For instance, consider the Petri net shown in Fig. 1(a). It has been discovered from the event log of the MAS with two *asynchronously* interacting agents. This model can reproduce event sequences of the original log, but its structure hides the valuable information on agent interactions.

[1] This work is supported by the Basic Research Program at the National Research University Higher School of Economics.

[⋆] Copyright © 2019 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0)

To overcome this problem, we suggest using a *compositional* approach to discover a MAS process model clearly indicating agents as subnets and their interactions through channels. Assume that we know in advance how channels are exploited (sending/receiving messages) by agents. The compositional process discovery is straightforward. Firstly, we discover process models of agents from filtered event logs. Then agent models are composed via channels. Consider the Petri net shown in Fig. 1(b). It can reproduce event sequences from the same event log as the model from Fig. 1(a). What is more important, this model explicitly indicates the agent behavior (left and right subnet) and the channels (gray nodes) used for interaction.



(a) (b)

Fig. 1. Discovering process models for multi-agent systems from event logs

Petri net composition has been extensively studied in the literature (e.g. in [4,8,13]). The main problem here is that composing correct models can result in a model with the incorrect behavior. We consider *soundness* (also referred to as *proper termination*) to be the key correctness property of process models.

In [6] we have proposed a compositional approach to discover process models of MAS with two asynchronously interacting agents. This approach involves *abstraction* to preserve soundness of agent models in their composition. Abstraction is implemented via special *morphisms* [7]. We abstract process models of agents regarding actions through which they exchange messages. A composition of abstract agent models is an interaction protocol (*interface*). Soundness of agent model composition results from the verified soundness of an interface.

However, the practical implementation of this approach may require extensive theoretical knowledge. Our work contributes to solving this problem in practice by applying *service interaction patterns* (SIPs). They have been described in [5]. SIPs provide generic solutions for designing composite services with several interacting entities. We have done the preliminary work on using simple patterns for compositional discovery of MAS process models in [16,17]. The practical value of a pattern-based approach has been justified by the experimental results.

In this study, we identify *typical* patterns describing asynchronous interaction on the basis of related research analysis. Patterns model agent interaction protocols at the abstract level. Following the approach proposed in [6], we also show these typical patterns are applied for compositional discovery of sound MAS models clearly indicating agent interactions.

Another view on the problem of discovering interactions from event logs has been discussed in [15], where artifact-centric approach to process mining has been proposed. The authors analyze life-cycles of data objects (artifacts) created and consumed during a business process execution.

The remainder of the paper is organized as follows. The next section provides an informal description of service interaction patterns. Section 3 recalls necessary notions from Petri net theory. Section 4 describes an approach to modeling and refining abstract service interaction patterns. In Section 5, we show experimental results on using patterns for mining MAS models from event logs. Section 6 concludes the paper by discussing results and possible continuations.

2 Service Interaction Patterns

Service interaction patterns provide a systematic approach to address the problem of organizing complex and large-scale interactions. They have been used in different contexts. Among the others, in [10,11] interaction patterns have also been explored within process modeling using BPMN. The important problem of pattern correctness has been discussed in [1,12], where patterns have been formalized using process algebras and open Petri nets. The authors used operating guidelines to define services interacting correctly with the given one.

Service interaction patterns are classified according to *the number of interacting entities:* (a) *bilateral* (two) and (b) *multilateral* (more than two). Also, service interaction patterns are classified w.r.t. *the way entities interact:* (a) *single transmission* patterns and (b) *multiple transmission* patterns [5]. The number of transmissions defines the number of times an agent can send (receive) a message to (from) the others.

In our work, we study bilateral patterns with both single and multiple transmissions. Table 1 provides a brief informal description of patterns considered in the paper. Short IDs are used to refer to these patterns in the text. Note that patterns SIP-1, SIP-2 and SIP-3 describe rather primitive interaction, since an agent sending a message is not supposed to get a response from another agent. More sophisticated communications are given in patterns SIP-4, SIP-5, SIP-6, when two agents actually exchange messages in different ways. SIP-7 is a multiple transmission pattern, where one agent decides to stop exchanging messages.

The aim of our work is to apply these patterns for compositional discovery of formal multi-agent system models from event logs. That is why, we show how to model these patterns using Petri nets in Section 4. Each pattern corresponds to a specification of a protocol according to which agents agree to communicate. Moreover, these patterns contains only *abstract* information on agent interaction providing minimal information on an internal agent behavior. We also describe

Table 1. Bilateral asynchronous interaction patterns

Pattern	**Short ID**	**Description**
Send (Receive)	SIP-1	An agent X sends (receives) a message to (from) an agent Y.
Concurrent Send (Receive)	SIP-2	An agent X concurrently sends (receives) several messages (>1) to (from) an agent Y.
Sending (Receiving) Choice	SIP-3	An agent X sends (receives) exactly one out of two (or more) alternative message sets to (from) an agent Y.
Send+Receive	SIP-4	An agent X sends a message to an agent Y. Subsequently, Y sends a response to X.
Concurrent Send+Receive	SIP-5	An agent X concurrently sends several messages (>1) to an agent Y. Then Y sends a response to each message received from X.
Sending+Receiving Choice	SIP-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 Send+Receive	SIP-7	The iterative implementation of SIP-4, s.t. message exchange process continues till an Agent X does not need responses from an Agent Y.

how to instantiate (*refine*) patterns with details of each agent behavior to obtain sound and structured models of multi-agent systems with two interacting agents.

3 Basic Notions

In this section, we recall definitions from Petri net theory necessary for constructing and refining formal models of service interaction patterns.

$\mathbb{N}$ denotes the set of non-negative integers. Let A be a set. The set of all finite non-empty sequences over A is denoted by A^+, and $A^* = A^+ \cup \{\epsilon\}$, where ϵ corresponds to the empty sequence. A function $m\colon A \to \mathbb{N}$ defines a *multiset* m over A. Let m_1, m_2 be a pair of multisets over A. The standard set operations are extended to multisets as well, i.e. (a) $m_1 \subseteq m_2 \Leftrightarrow m_1(a) \leq m_2(a)$, (b) $m' = m_1 \cup m_2 \Leftrightarrow m'(a) = m_1(a) + m_2(a)$ and (c) $m'' = m_1 \setminus m_2 \Leftrightarrow m''(a) = max(0, m_1(a) - m_2(a))$ for all $a \in A$.

A *Petri net* is a triple $N = (P, T, F)$, where where P and T are two disjoint sets of places and transitions, i.e. $P \cap T = \varnothing$, and $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation, where $dom(F) \cup cod(F) = P \cup T$. Graphically, places are shown by circles, transitions — by boxes, and flow relation — by arcs.

Let $N = (P, T, F)$ be a Petri net, and $X = P \cup T$. The set ${}^\bullet x = \{y \in X | (y, x) \in F\}$ denotes the *preset* of $x \in X$. The set $x^\bullet = \{y \in X | (x, y) \in F\}$ denotes the *postset* of $x \in X$. Let $A \subseteq X$, then ${}^\bullet A = \bigcup_{x \in A} {}^\bullet x$, $A^\bullet = \bigcup_{x \in A} x^\bullet$. By $N(A)$ we denote a *subnet* of N *generated by* A, i.e. $N(A) = (P \cap A, T \cap A, F \cap (A \times A))$. Note that we consider nets, s.t. $\forall t \in T\colon |{}^\bullet t| \geq 1$ and $|t^\bullet| \geq 1$.

A *marking* (state) of a Petri net $N = (P, T, F)$ is a function $m \colon P \to \mathbb{N}$. A marking is shown by putting $m(p)$ black dots (tokens) inside a place $p \in P$. A *marked* Petri net $N = (P, T, F, m_0)$ is a Petri net together with its initial marking m_0. A marking m *enables* a transition $t \in T$, denoted $m[t\rangle$, if ${}^\bullet t \subseteq m$. The *firing* t at m leads 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* of $N = (P, T, F, m_0)$ if $w = t_1 t_2 \dots t_n$ and $m_0[t_1\rangle m_1[t_2\rangle \dots m_{n-1}[t_n\rangle m_n$. Then we can write $m_0[w\rangle m_n$. The set of all firing sequences of N is denoted by $FS(N)$.

A marking m of $N = (P, T, F, m_0)$ is *reachable* if $\exists w \in FS(N) \colon m_0[w\rangle m$. Any reachable marking is reachable from itself, i.e. $m[\epsilon\rangle m$. The set of all markings reachable from m is denoted by $[m\rangle$. A reachable marking is *dead* if it does not enable any transition. N is *safe* if $\forall p \in P \, \forall m \in [m_0\rangle \colon m(p) \leq 1$. In other words, in a safe net N we have $\forall m \in [m_0\rangle \colon m \subseteq P$.

A *state machine* is a connected Petri net $N = (P, T, F)$, s.t. $\forall t \in T \colon |{}^\bullet t| = |t^\bullet| = 1$. A subnet of a marked Petri net $N = (P, T, F, m_0)$ identified by a subset of places $A \subseteq P$ and its neighborhood, i.e. $N(A \cup ({}^\bullet A^\bullet))$, 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. Then N is *state machine decomposable* (SMD).

Workflow nets form a special subclass of Petri nets used for modeling processes. They have an explicitly specified initial and final state. The initial state is obviously to correspond with the initial marking. We define *generalized workflow nets* which initial and final states are expressed in terms of subsets of places. Note that SMD GWF-nets are safe.

A marked Petri net $N = (P, T, F, m_0, m_f)$ is a generalized workflow net (GWF-net) if and only if:

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

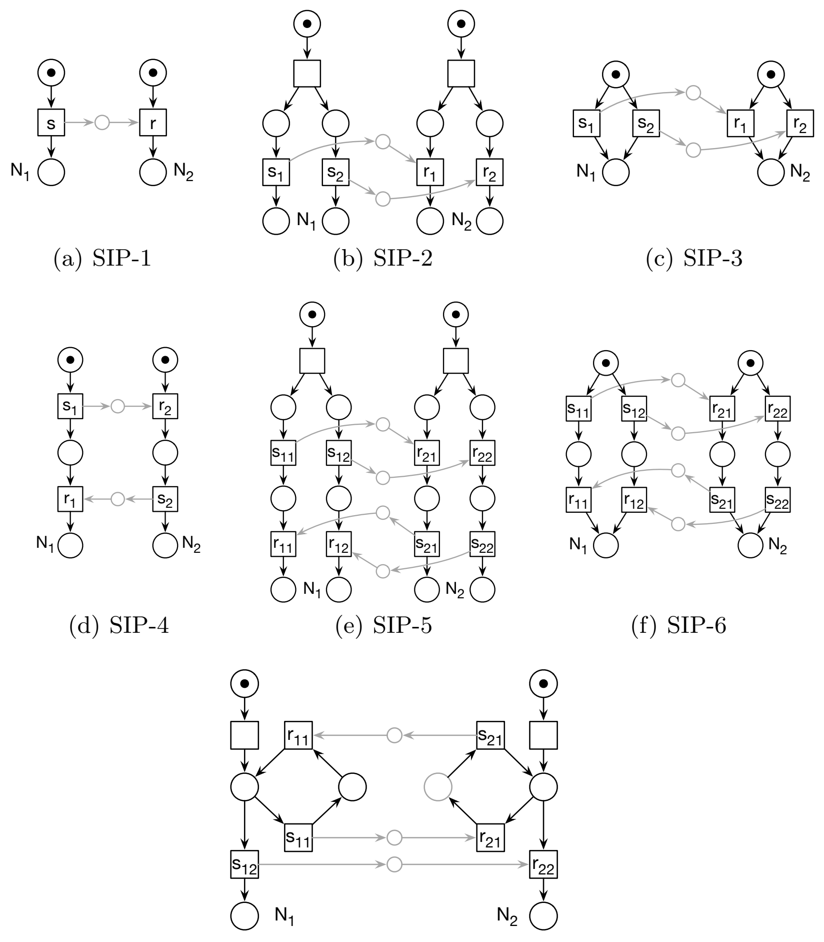
The third requirement intuitively means that each node of a GWF-net should lie on a path from a place in the initial state to a place in the final state. The correctness of processes modeled via GWF-nets is considered in terms of their *soundness*. A GWF-net $N = (P, T, F, m_0, m_f)$ is *sound* if and 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. $\forall t \in T \, \exists m \in [m_0\rangle \colon m[t\rangle$.

4 Modeling and Refining Abstract Service Interaction Patterns

Figure 2 shows seven sound and state machine decomposable GWF-nets constructed according to the specification of *abstract* patterns given in Section 2.

Each GWF-net is also a *channel-composition* of two disjoint GWF-nets. Channels are places which we add to connect selected transitions of GWF-nets. Channels model message exchange between two agents. A channel-composition of two GWF-nets N_1 and N_2 via a set of channels C is denoted by $N_1 \oplus_C N_2$, where C is a parameter. In an abstract pattern $N_1 \oplus_C N_2$, N_1 and N_2 are abstract models of agent behavior. The precise definition of the channel-composition has been given in [6]. Channels are indicated as small gray places in Fig. 2.



(a) SIP-1 (b) SIP-2 (c) SIP-3

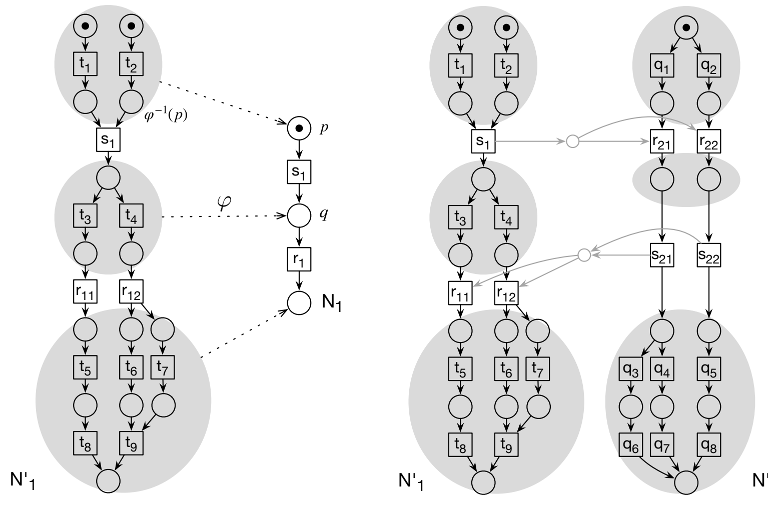
(d) SIP-4 (e) SIP-5 (f) SIP-6

(g) SIP-7

Fig. 2. Channel-composed GWF-nets modeling bilateral interaction patterns

We *refine* GWF-nets of abstract interaction patterns with detailed models of agent behavior following the compositional approach described in [6]. Given a channel-composed GWF-net of an abstract pattern $N_1 \oplus_C N_2$, N_1 and N_2 are refined with agent behavior details. Refinement is implemented with the help of α-morphisms [7].

The α-morphism is a mapping between two state machine decomposable GWF-nets: *from* a refined model *to* its abstraction. The α-morphism is a total surjective mapping. It maps nodes of a refined model onto nodes of its abstraction. Further we denote the α-morphism between two GWF-nets by $\varphi\colon N_1 \to N_2$, where N_2 is an abstract model, and N_1 is its refinement. Consider the α-morphism $\varphi\colon N_1' \to N_1$ shown in Fig. 3(a), where N_1' is a refinement of N_1 from the abstract pattern SIP-4. The refinement of places is depicted by shaded ovals and by the transition labels explicitly, which can also result in splitting transitions of an abstract model. As shown in Fig. 3(a), the transition r_1 of the abstract GWF-net N_1 is refined (split) by a pair of transitions r_{11} and r_{12} of the detailed GWF-net N_1', whereas the transition s_1 is not refined.



(a) $\varphi\colon N_1' \to N_1$ (b) $N_1' \oplus_C N_2'$, $|C| = 2$

Fig. 3. Refining abstract interaction pattern SIP-4

Note that α-morphisms allow us to substitute places of an abstract net N_2 with *acyclic* subnets of a detailed net N_1 (under $\varphi\colon N_1 \to N_2$). On the one hand, when a transition of N_1 is mapped to a transition of N_2, neighborhoods of these transitions should also be related by the α-morphism. In Fig. 3(a), the transition r_{11} of N_1' is mapped to the transition r_1 of N_1. Correspondingly, the image of the input place of r_{11} (under φ) is the input place of r_1. On the other hand, when a transition of N_1 is mapped to a place of N_2, its neighborhood is also mapped to this place. In Fig. 3(a), the transition t_9 is mapped to the final place of N_1. Then its neighborhood is also mapped to the final place of N_2.

The main motivation behind α-morphisms is the ability to ensure that properties of an abstract model hold in its refinement (refer to Lemma 1 in [7]). For instance, when the transition s_1 of N_1 from Fig. 3(a) fires, the token is moved to the place q. Correspondingly, when the transition s_1 of N_1' fires, no tokens are left in the subnet $\varphi^{-1}(p)$ of N_1' refining the place p of N_1.

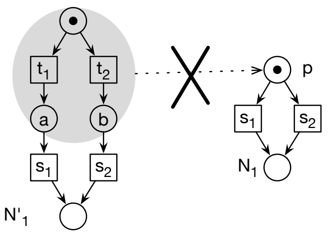
According to the main result of [6], in the general case one can simultaneously refine N_1 and N_2 in their channel-composition $N_1 \oplus_C N_2$ by N_1' and N_2', if there are two corresponding α-morphisms $\varphi_i \colon N'_i \to N_i$ for $i = 1, 2$, obtaining $N'_1 \oplus_C N'_2$ as a result. Moreover, if $N_1 \oplus_C N_2$, N'_1 and N'_2 are sound GWF-nets, then $N'_1 \oplus_C N'_2$ is also a sound GWF-net.

Thus, we can refine interaction patterns with sound GWF-nets corresponding to the detailed models of agent behavior. As a result, we obtain sound GWF-nets of multi-agent systems with two asynchronously interacting agents. Therefore, each pattern defines a class of sound process models for multi-agent systems.

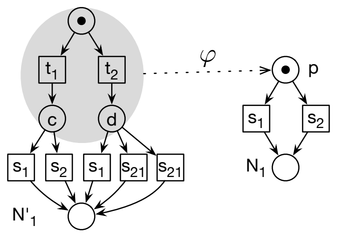
For example, consider the pattern SIP-4 (see Fig. 2(d)). Its refinement is provided in Fig. 3(b), according to α-morphisms $\varphi_1 \colon N'_1 \to N_1$ given in Fig. 3(a) and $\varphi_2 \colon N'_2 \to N_2$ indicated by shaded ovals and transition labels. A possible refinement of the pattern SIP-7 has been given in [6] (see Fig. 5(a) there).

Let us consider the patterns with conflicts (SIP-3, SIP-6 and SIP-7) in more detail. A set of transitions is in conflict if the share at least one common input place. Conflicts of the abstract model should be properly refined (given by definition of α-morphisms in [7]). This is clarified in the following example.

Consider refinements of N_1 from the pattern SIP-3 shown in Fig. 4. The refinement shown in Fig. 4(a) is incorrect, since the output places (a, b) of the shaded subnet have only one outgoing transition, whereas the abstract place p has the choice between two transitions. The refinement shown in Fig. 4(b) is valid, since the output places (c, d) of the shaded subnet has "the same choices" the abstract place p does. Moreover, in the case of the place d, the transition s_2 is split into two other transitions (s_{21} and s_{22}) at the detailed level in N'_1.



(a) wrong refinement



(b) correct refinement

Fig. 4. Refining conflicts of abstract interaction patterns

5 Mining Structured and Sound Multi-Agent System Models: Experimental Evaluation

In this section, we show the experimental results on applying interaction patterns described earlier to process discovery within multi-agent systems. We have compared quality of models synthesized from event logs of multi-agent systems in two ways: directly and by means of composition with interaction patterns. In our experiments *inductive miner* [14] has been used, since it guarantees *soundness* and *state machine decomposability* (see Section 3) of discovered models.

5.1 Main Layout of Experiments

The experiments have been conducted according to the plan provided below.

Step 1. Take a service interaction pattern which is modeled in terms of the channel-composition $N_1 \oplus_C N_2$ and refine it by using α-morphisms $\varphi_i \colon N_i' \to N_i$ $(i = 1, 2)$. Thus, obtain a sound system model $N_1' \oplus N_2'$.
Step 2. Compute an event log L of $N_1' \oplus N_2'$ by simulating it using the tool presented in [18]. This event log is considered to be the main input to the process discovery algorithm.
Step 3. Discover a GWF-net N_d from the event log L *directly*.
Step 4. Filter the event log L according to the behavior of individual agents obtaining the two sub-logs $L_{N_1'}$ and $L_{N_2'}$.
Step 5. Discover two sound GWF-nets $\widetilde{N}_1'$ and $\widetilde{N}_2'$ from the sub-logs $L_{N_1'}$ and $L_{N_2'}$ computed at the previous step.
Step 6. Having constructed the α-morphism $\widetilde{\varphi}_i \colon \widetilde{N}_i' \to N_i$ $(i = 1, 2)$, get a *compositionally* discovered GWF-net $N_c = \widetilde{N}_1' \oplus_C \widetilde{N}_2'$.
Step 7. Compare quality of N_c with that of N_d constructed at Step 4.

The construction of the two α-morphisms at Step 6 is done manually so far. An algorithm for constructing α-morphisms is subject for further investigations.

There are four main quality dimensions used in process discovery: *fitness*, *precision*, *simplicity* and *generalization* [9]. In our experiments, we estimate precision and simplicity (Step 7). Precision shows how much extra behavior a discovered model adds in comparison with that given in an initial event log.

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

- the number of places, transitions and arcs;
- the number of neighboring transitions between different agents.

The following paragraph explains the main idea behind the notion of neighboring transitions.

Neighboring transitions. Inductive miner produces a sound GWF-net $N = (P, T, F, m_0, m_f)$ with transitions labeled by a transition labeling function $\lambda \colon T \to A \cup \{\tau\}$. A is a set of *visible* event names recorded in an event log from which N is discovered, whereas τ is a special label of a *silent* transitions. Silent transitions do not correspond to any event from an event log.

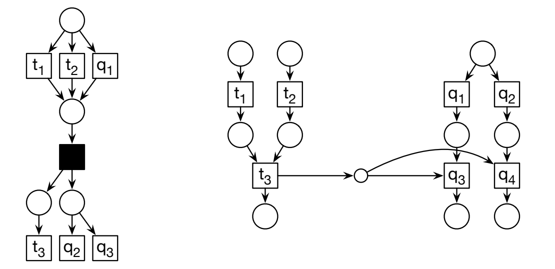
We introduce the notion of neighboring transitions as an attempt to measure the extent to which a structure of a discovered model correspond to the actual multi-agent system structure w.r.t. agent interaction. In other words, we expect a structured model of a multi-agent system to explicitly indicate behavior of individual agents as well as the way the communicate by sending/receiving messages (via channels). Below we give precise definitions.

Let $N = (P, T, F, m_0, m_f)$ be a GWF-net, and $\lambda \colon A \cup \{\tau\}$ be a transition labeling function. Two transitions $t_1, t_2 \in T$, s.t. $\lambda(t_1) \neq \tau$ and $\lambda(t_2) \neq \tau$, are called *neighboring* iff there exists a path in N connecting t_1 and t_2, where other transitions are silent. This is expressed symbolically as follows:

- $(t_1, t_2) \in F^*$, where F^* is the reflexive transitive closure of F, and
- $\forall t \in T \setminus \{t_1, t_2\} \colon ((t_1, t) \in F^* \wedge (t, t_2) \in F^*) \Rightarrow \lambda(t) = \tau$.

When N represents a model for a multi-agent system with two asynchronously interacting agents X and Y, $T = T_X \cup T_Y$, s.t. $T_X \cap T_Y = \varnothing$. We are interested in finding neighboring transition pairs involving *different* agents. Nb_N denotes the number of neighboring transition pairs of N from $(T_X \times T_Y) \cup (T_Y \times T_X)$, s.t. two symmetric pairs are counted as a single pair. Intuitively, the bigger the value of Nb_N is, the less transparent and understandable the structure N is w.r.t. agent interaction. Nb_N of a "perfect" model of a multi-agent system is minimized up to transitions connected directly via channel places.

Consider two GWF-net fragments shown in Fig. 5, where t_i and q_i transition labels correspond to actions of different agents. Silent transitions are indicated by black boxes. The fragment shown in Fig. 5(a) has 5 neighboring transitions pairs, whereas Nb_N of the fragment shown in Fig. 5(b) is 2 corresponding to the only transitions connected via the small channel place.



(a) $\text{Nb}_N = 5$ (b) $\text{Nb}_N = 2$

Fig. 5. Neighboring transitions

5.2 Experiment Results

Table 2 provides characteristics of event logs artificially prepared for experimental evaluation. The number of traces has been fixed to 5000. Table 3 presents the quality comparison of models for multi-agent systems discovered directly (Step 2) and compositionally (Steps 5-7) from a set of prepared event logs (Step 1).

Table 2. Characteristics of artificially generated event logs

	SIP-1	**SIP-2**	**SIP-3**	**SIP-4**	**SIP-5**	**SIP-6**	**SIP-7**
Number of events	67366	115078	59851	110000	157532	81279	76889
Min events per trace	12	22	11	22	31	15	6
Max events per trace	15	24	13	22	32	17	97
Avg events per trace	13	23	12	22	32	16	15

The following conclusions can be made based on the results obtained:

1. Compositionally discovered models are more compact w.r.t. the number of nodes and arcs;
2. Precision of composed models is generally higher in comparison with the precision of directly discovered models (except for SIP-2 and SIP-4);

Table 3. Quality evaluation of models for multi-agent systems

	SIP-1	**SIP-2**	**SIP-3**	**SIP-4**	**SIP-5**	**SIP-6**	**SIP-7**
			Direct discovery				
Number of places	35	59	42	49	64	72	29
Number of transitions	41	50	46	46	69	84	34
Number of arcs	98	140	112	120	170	198	78
Nb_N	22	95	25	61	85	117	22
Precision	0,6890	0,5646	0,7768	0,6733	0,4554	0,6586	0,6495
			Compositional discovery				
Number of places	33	55	35	46	69	66	29
Number of transitions	32	49	35	39	57	72	28
Number of arcs	76	133	86	102	155	157	72
Nb_N	4	6	6	4	7	14	12
Precision	0,8017	0,4477	0,8162	0,6745	0,4342	0,8414	0,8500

3. Using compositional approach results in minimizing Nb_N up to transitions connected only via channel places.

In the case of patterns SIP-2 and SIP-4, precision decrease stems from the structure of the agent models and the lack of event data. Compositionally discovered models with concurrent branches allow for more behavior in comparison with directly discovered models. However, precision of these models will grow, if event logs have more examples (more traces) of the possible observed behavior.

6 Conclusion

This paper deals with the problem of discovering structured and sound models of multi-agent systems from their event logs. The proposed approach is based on using asynchronous service interaction patterns. A system model is composed from two agent models w.r.t. an interaction pattern. We have constructed formal models of seven typical interaction patterns. They describe communication between two agents at the *abstract* level. By using channel-composition and α-morphisms as described in [6], we show how abstract pattern models can be *refined* to obtain sound models of multi-agent systems.

We have conducted the series of experiments on using the proposed interaction patterns for compositional process discovery. Experiment results show that composition allows us to obtain more compact and precise models. Moreover, the structure of compositionally discovered models explicitly indicates agents as subnets and their interaction as channel places. To quantify it, we have introduced the notion of *neighboring transitions*. The number of neighboring transitions in composed models is exactly the number of transitions connected via channels.

The future research will be focused on working with more general multilateral patterns involving k (more than two) interaction agents. We also plan to conduct more experiments applying other process discovery algorithms provided that discovered models meets necessary requirements. Note also that formal models of the proposed patterns can be regularly composed (sequencing, alternative of parallel composition) producing more complex interaction patterns.

References

1. van der Aalst, W.M.P., Mooij, A., Stahl, C., Wolf, K.: Service interaction: Patterns, formalization, and analysis. In: SFM 2009. LNCS, vol. 5569, pp. 42–88. Springer, Heidelberg (2009)
2. van der Aalst, W.: Process Mining - Data Science in Action. Springer, 2 edn. (2016)
3. Augusto, A., Conforti, R., Dumas, M., Rosa, M.L., Maggi, F.M., Marrella, A., Mecella, M., Soo, A.: Automated discovery of process models from event logs: Review and benchmark. IEEE TKDE (2018)
4. Baldan, P., Corradini, A., Ehrig, H., Heckel, R.: Compositional modeling of reactive systems using open nets. In: CONCUR 2001. LNCS, vol. 2154, pp. 502–518. Springer (2001)
5. Barros, A., Dumas, M., ter Hofstede, A.: Service interaction patterns. In: BPM 2005. LNCS, vol. 3649, pp. 302–318. Springer, Heidelberg (2005)
6. Bernardinello, L., Lomazova, I., Nesterov, R., Pomello, L.: Compositional discovery of workflow nets from event logs using morphisms. In: Proceedings of ATAED-2018. CEUR Workshop Proceedings, vol. 2115, pp. 23–38. CEUR-WS.org (2018)
7. Bernardinello, L., Mangioni, E., Pomello, L.: Local state refinement and composition of elementary net systems: An approach based on morphisms. In: ToPNoC VIII. LNCS, vol. 8100, pp. 48–70. Springer, Heidelberg (2013)
8. Best, E., Devillers, R., Hall, J.G.: The box calculus: A new causal algebra with multi-label communication. LNCS, vol. 609, pp. 21–69. Springer (1992)
9. Buijs, J.C.A.M., van Dongen, B.F., van der Aalst, W.M.P.: On the role of fitness, precision, generalization and simplicity in process discovery. In: OTM 2012. pp. 305–322. Springer, Heidelberg (2012)
10. Campagna, D., Kavka, C., Onesti, L.: Bpmn 2.0 and the service interaction patterns: Can we support them all? In: ICSOFT 2014. CCIS, vol. 555, pp. 3–20. Springer, Heidelberg (2015)
11. Decker, G., Barros, A.: Interaction modeling using bpmn. In: BPM 2007 Workshops. LNCS, vol. 4928, pp. 208–219. Springer, Heidelberg (2008)
12. Decker, G., Puhlmann, F., Weske, M.: Formalizing service interactions. In: BPM 2006. LNCS, vol. 4102, pp. 414–419. Springer, Heidelberg (2006)
13. Girault, C., Valk, R.: Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications. Springer, Heidelberg (2003)
14. Leemans, S., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from event logs - a constructive approach. In: PETRI NETS 2013. LNCS, vol. 7927, pp. 311–329. Springer, Heidelberg (2013)
15. Lu, X., Nagelkerke, M., v. d. Wiel, D., Fahland, D.: Discovering interacting artifacts from erp systems. IEEE Transactions on Services Computing 8(6), 861–873 (2015)
16. Nesterov, R.A., Lomazova, I.A.: Using interface patterns for compositional discovery of distributed system models. Proceedings of the Institute for System Programming 29(4), 21–38 (2017)
17. Nesterov, R.A., Lomazova, I.A.: Compositional process model synthesis based on interface patterns. In: TMPA 2017. CCIS, vol. 779, pp. 151–162. Springer (2018)
18. Nesterov, R.A., Mitsyuk, A., Lomazova, I.A.: Simulating behavior of multi-agent systems with acyclic interactions of agentss. Proceedings of the Institute for System Programming 30(3), 285–302 (2018)
19. Reisig, W.: Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies. Springer (2013)