Formal Modeling and Discovery of Multi-instance Business Processes: A Cloud Resource Management Case Study

Cong Liu, *Member, IEEE*

***Abstract*—Process discovery, as one of the most challenging process analysis techniques, aims to uncover business process models from event logs. Many process discovery approaches were invented in the past twenty years; however, most of them have difficulties in handling multi-instance sub-processes. To address this challenge, we first introduce a multi-instance business process model (MBPM) to support the modeling of processes with multiple sub-process instantiations. Formal semantics of MBPMs are precisely defined by using multi-instance Petri nets (MPNs) that are an extension of Petri nets with distinguishable tokens. Then, a novel process discovery technique is developed to support the discovery of MBPMs from event logs with sub-process multi-instantiation information. In addition, we propose to measure the quality of the discovered MBPMs against the input event logs by transforming an MBPM to a classical Petri net such that existing quality metrics, e.g., fitness and precision, can be used. The proposed discovery approach is properly implemented as plugins in the ProM toolkit. Based on a cloud resource management case study, we compare our approach with the state-of-the-art process discovery techniques. The results demonstrate that our approach outperforms existing approaches to discover process models with multi-instance sub-processes.**

***Index Terms*—Cloud resource management process, multi-instance Petri nets (MPNs), multi-instance sub-processes, process discovery, quality evaluation.**

I. Introduction

PROCESS mining aims at extracting process-related insights from business process event logs [1], [2]. Process discovery, as one of the most difficult process mining tasks, has received a lot of attention in the past years. Various discovery techniques that take as input event logs and produce process models have been proposed, e.g., alpha miner [3], heuristic miner [4], and inductive miner [5]. However, existing process discovery techniques cannot be directly used when an event log contains information of multi-instance sub-processes. To understand multi-instance sub-processes, we first explain the notion of sub-processes. Consider a business process out-sourcing scenario where one organization sub-contracts part of its businesses to another organization [6]. The sub-contracted process is regarded as a sub-process of the parent (original) process. Multi-instance sub-processes mean that multiple concurrent instances of a sub-process are executed within the same parent process instantiation [7]. Consider for example an online shopping scenario, multiple delivery sub-processes can be instantiated in parallel when a consumer confirms an order of multiple products with different delivery addresses. In this case, the delivery process is a multi-instance sub-process of the order process.

The notion of multi-instance sub-process imposes new challenges for existing techniques to discover high-quality models. To address this challenge, we present a novel technique to discover multi-instance business process models from event logs with sub-process multi-instantiation information. The main contributions of this paper are summarized as follows:

1) We introduce multi-instance business process models (MBPMs) to support the modeling of processes with multiple sub-process instantiations;

2) We present an extension of classical Petri nets, called multi-instance Petri nets (MPNs), to formalize the semantics of MBPMs; and

3) We present a novel process discovery technique to support the discovery of MBPMs from event logs with sub-process multi-instantiation information.

The rest of this paper is organized as follows. Section II reviews some related work. Section III introduces a motivating example, based on which we showcase the main challenges. Section IV reviews some preliminaries. Then, Section V formalizes the multi-instance Petri nets. Section VI shows how to discover MBPMs from event logs. Section VII presents tool support. Section VIII performs the experimental evaluation. Finally, Section IX concludes the paper.

II. Related Work

Process discovery can be used to reconstruct business process models from event logs [1]. As an often cited example, the alpha miner defines four kinds of ordering relations among activities, including directly-follow relation, choice relation, concurrency relation, and causality relation, based on which a Petri net is constructed to describe the behavior recorded in

Manuscript received July 10, 2020; revised October 8, 2020; accepted December 6, 2020. This work was supported by the National Natural Science Foundation of China (61902222), the Taishan Scholars Program of Shandong Province (tsqn201909109), the Natural Science Excellent Youth Foundation of Shandong Province (ZR2021YQ45), and the Youth Innovation Science and Technology Team Foundation of Shandong Higher School (2021KJ031). Recommended by Associate Editor Shouguang Wang.

Citation: C. Liu, "Formal modeling and discovery of multi-instance business processes: A cloud resource management case study," *IEEE/CAA J. Autom. Sinica*, vol. 9, no. 12, pp. 2151–2160, Dec. 2022.

The author is with the School of Computer Science and Technology, Shandong University of Technology, Zibo 255000, and also with the College of Computer Science and Engineering, Shandong University of Science and Technology, Qingdao 266590, China (e-mail: liucongchina@sdust.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JAS.2022.106109

the event log [3]. Several improvements on the alpha miner are made to deal with short loops [8], non-free-choice constructs [9], etc. To handle noise and infrequent behavior, Weijters *et al*. [4] introduce the heuristic miner by considering the frequency of each dependency relation, which is robust to noise. More recently, inductive miner [5], as the state-of-the-art process discovery algorithm, is proposed to handle both noisy and incomplete event logs.

Different from majority discovery algorithms that generate flat process models, Conforti *et al.* [10] propose to mine BPMN models with multi-instance markers and sub-process information. However, this approach relies heavily on the primary and foreign key attributes to obtain inter-relations between parent processes and sub-processes as well as multi-instance markers. Another related area is the artifact-centric process discovery that focuses on the discovery of artifact lifecycle models and their interactions [11]. An artifact describes the lifecycle of a business object (e.g., a purchase order). van Eck *et al.* [12] propose to discover both artifact lifecycle models and interactions among them. However, an artifact lifecycle model is usually represented as a flat model which cannot handle the hierarchical behavior properly. To formalize semantics of BPMN, a recursive ECATNet is introduced in [13]. Based on the recursive ECATNet, typical BPMN features, e.g., cancellation, multiple instantiation of sub-processes, and exception handling, are supported. More recently, we propose to discover hierarchical Petri nets from event logs by identifying sub-processes in [14] and [15]. However, the multi-instance feature of sub-processes is not supported.

Another related research domain is software process mining. The main focus is to distil behavioral models from software execution traces. A software behavioral model is typically expressed as process models with hierarchical (or nested) sub-processes [16], [17]. The discovery of sub-processes depends on the calling relations among method calls that are software specific and not available in a more general case. More recently, Leemans *et al.* [18] propose an integrated tool that aims to discover hierarchical process models from software execution logs. However, these approaches [16]−[18] do not support the discovery of multi-instance sub-processes.

III. Motivating Example

Cloud computing is a new paradigm that provides on-demand computing resources, e.g., servers and platforms, to users [19]. Netflix Asgard[1] is an open-source cloud resource management tool on Amazon Web Services. Consider for example its rolling upgrade process. This upgrade process replaces a group of virtual machines at one time on Amazon Web Services. Assume that an application is running on 3 virtual machines, and these virtual machines are with version v1. Now we need to upgrade and replace all these virtual machines with version v2. This is done by taking the 3 virtual machines running version v1 out of service and replacing them with 3 virtual machines running version v2.

The whole upgrading process is described as Fig. 1. It starts with four activities (SRU (start rolling upgrade task), ULC (update launch configuration), SOI (sort instances) and GSI (get status information)) to prepare the upgrading process. Then, activity UPI (upgrading instances) is instantiated to invoke three sub-process instances that each upgrades a virtual machine with four activities (DI (deregister an old instance), TI (terminate an old instance), SI (start a new instance), and RI (register a new instance)). These three upgrading sub-processes are executed in parallel. After all sub-process instances have been finished, the process completes with CRU (rolling upgrade task completed).

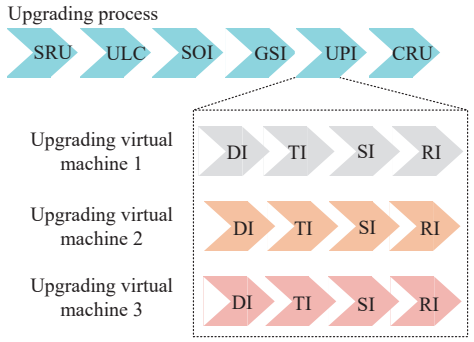

Fig. 1. Upgrading process.

During execution of the upgrading process, event logs are recorded. Note that the multi-instance sub-processes typically generate events that are interleaved with the events generated from the parent process in the resulting sequential event logs. Table I shows an event log that is collected during one execution of the upgrading process. Each row refers to a recorded event and an activity execution is recorded as two independent events representing its start and completion.

By examining these event logs, we aim to 1) understand how the upgrading process is actually executed; and 2) check if the actual execution follows the pre-defined process. To this end, we need 1) a suitable formalization model to precisely describe the behavior of underlying processes, especial for the multi-instance sub-process behavior; and 2) an effective technique to discover such models from interleaved event logs.

IV. Preliminaries

Let $\mathbb{N}=\{0,1,2,\ldots\}$ be the natural number set. A multi-set over S is defined as $m: S \to \mathbb{N}$ where for any $s \in S$, $m(s)$ is the multiplicity of s, e.g., $m=[p^3,q^2]$ is a multi-set over $S=\{p,q\}$ where $m(p)=3$ and $m(q)=2$. The set of all multi-sets over S is denoted by $\mathbb{N}^S$. The powerset of S is denoted by $\mathcal{P}(S)=\{S' | S' \subseteq S\}$. The set of all finite sequences over set S is denoted by S^*. $f: X \to Y$ is a function, i.e., $dom(f)=X$ is the domain and $rng(f)=\{f(x) | x \in dom(f)\} \subseteq Y$ is the range.

Definition 1: Let X be a set and $Q \subseteq X$ be its subset. $\restriction_Q \in X^* \to Q^*$ is a projection function and is defined recursively: $\langle\,\rangle \restriction_Q = \langle\,\rangle$; and for $\sigma \in X^*$ and $x \in X$:

$$(\langle x \rangle \circ \sigma) \restriction_Q = \begin{cases} \sigma \restriction_Q, & \text{if } x \notin Q \\ \langle x \rangle \circ (\sigma \restriction_Q), & \text{if } x \in Q. \end{cases} \tag{1}$$

A. Petri Nets and Workflow Nets

Petri nets are widely used to model and analyze business

[1] https://github.com/Netflix/asgard

TABLE I
A FRAGMENT OF EVENT LOG

Case	Event	Activity	Timestamp	Transaction	Virtual machine
1	e_1	SRU	04-05-2014 09:30	start	–
1	e_2	SRU	04-05-2014 09:32	complete	–
1	e_3	ULC	04-05-2014 09:34	start	–
1	e_4	ULC	04-05-2014 09:38	complete	–
1	e_5	SOI	04-05-2014 09:41	start	–
1	e_6	SOI	04-05-2014 09:45	complete	–
1	e_7	GSI	04-05-2014 09:46	start	–
1	e_8	GSI	04-05-2014 09:49	complete	–
1	e_9	UPI	04-05-2014 09:50	start	–
1	e_{10}	DI	04-05-2014 09:56	start	VM1
1	e_{11}	DI	04-05-2014 09:57	start	VM2
1	e_{12}	DI	04-05-2014 09:58	start	VM3
1	e_{13}	DI	04-05-2014 10:08	complete	VM1
1	e_{14}	DI	04-05-2014 10:10	complete	VM2
1	e_{15}	TI	04-05-2014 10:11	start	VM1
1	e_{16}	DI	04-05-2014 10:15	complete	VM3
1	e_{17}	TI	04-05-2014 10:12	start	VM2
1	e_{18}	TI	04-05-2014 10:28	complete	VM1
1	e_{19}	TI	04-05-2014 10:30	complete	VM2
1	e_{20}	SI	04-05-2014 10:31	start	VM1
1	e_{21}	SI	04-05-2014 10:33	start	VM2
1	e_{22}	TI	04-05-2014 10:36	start	VM3
1	e_{23}	TI	04-05-2014 10:56	complete	VM3
1	e_{24}	SI	04-05-2014 11:12	complete	VM1
1	e_{25}	RI	04-05-2014 11:16	start	VM1
1	e_{26}	SI	04-05-2014 11:22	complete	VM2
1	e_{27}	RI	04-05-2014 11:26	complete	VM1
1	e_{28}	RI	04-05-2014 11:28	start	VM2
1	e_{29}	SI	04-05-2014 11:30	start	VM3
1	e_{30}	RI	04-05-2014 11:40	complete	VM2
1	e_{31}	SI	04-05-2014 11:48	complete	VM3
1	e_{32}	RI	04-05-2014 11:53	start	VM3
1	e_{33}	RI	04-05-2014 12:12	complete	VM3
1	e_{34}	UPI	04-05-2014 12:20	complete	–
1	e_{35}	CRU	04-05-2014 12:25	start	–
1	e_{36}	CRU	04-05-2014 12:30	complete	–

processes [20]–[25] emergency response processes [26], [27], manufacturing processes [28], etc. Basic concepts and notations on Petri nets are reviewed following [29].

Definition 2: A Petri net is a 4-tuple $PN = (P, T, F, l)$, satisfying:

1) P is a finite set of places and T is a finite set of transitions where $P \cap T = \emptyset$;

2) $F \subseteq (P \times T) \cup (T \times P)$ is a set of directed arcs, called flow relation; and

3) $l : T \to \mathcal{A}$ is a labeling function where $\mathcal{A}$ is a set of labels and $\tau \in \mathcal{A}$ denotes the invisible label.

Given a $PN = (P, T, F, l)$, we define the preset and postset of transitions and places. For each $x \in P \cup T$, ${}^\bullet x = \{y | (y, x) \in F\}$ is the preset and $x^\bullet = \{y | (x, y) \in F\}$ is the postset. We use markings to describe the semantics of a Petri net. A marking $m \in \mathbb{N}^P$ is a multiset of places, which indicates how many tokens each place contains. (PN, m_0) is a marked net where m_0 is its initial marking. Fig. 2 shows a marked Petri net example. [source] is its initial marking. A transition $t \in T$ is enabled in marking m, denoted as $(PN, m)[t\rangle$ if $\forall p \in {}^\bullet t : m(p) \geq 1$. Consider the example Petri net with $m = [p_3, p_4]$, transition d is enabled, i.e., $(PN, m)[d\rangle$. An enabled transition t may fire and

result in a new marking m' with $m' = (m \setminus {}^\bullet t) \cup t^\bullet$, denoted by $(PN, m)[t\rangle(PN, m')$, i.e., one token is removed from each of its preset and one token is added for each of its postset. By firing transition d under marking $m = [p_3, p_4]$, we have $(PN, [p_3, p_4])[d > (PN, [p_5])$ for the example Petri net.

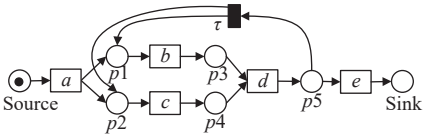

Fig. 2. A labeled Petri net example.

In this paper, we consider a special type of Petri net which is known as workflow net (WF-net) [30]. A WF-net is a Petri net with a dedicated source place where the process starts and a dedicated sink place where the process ends, and all nodes are on a path from source to sink.

Definition 3: Let $PN = (P, T, F, l)$ be a labeled Petri net and $\bar{t}$ be a transition not in T. PN is a WF-net if:

1) P contains one source place i such that ${}^\bullet i = \emptyset$;

2) P contains one sink place o such that $o^\bullet = \emptyset$;

3) $l(\bar{t}) = \tau$; and

4) $\overline{PN} = (P, T \cup \{\bar{t}\}, F \cup \{(o, \bar{t}), (\bar{t}, i)\})$ is strongly connected, i.e., there is a directed arc between any pair of nodes in $\overline{PN}$.

$\overline{PN}$ is referred to as the short-circuited net. The unique sink place o is connected to the unique source place i in $\overline{PN}$. A WF-net is sound if and only if its corresponding short-circuited net is 1-bounded and live [30]. Fig. 2 shows an example of sound WF-net with $i = source$ and $o = sink$.

B. Event Logs

An event log is composed of a finite set of events with various attributes, e.g., activity, timestamp, and transaction (or lifecycle) information. In the following, standard attributes of events are defined.

Definition 4: Let ξ be the event universe, i.e., the set of all possible event identifiers. Events may have various attributes. For any $e \in \xi$, $\#_n(e)$ is the value of attribute n for event e.

An activity may take time and have transactional states, e.g., schedule, start, suspend and complete. Note that we only consider the start and complete states, i.e., events are recorded at the moment an activity is started and completed. For an arbitrary event $e \in \xi$, the following attributes are involved:

1) $\#_{\text{case}}(e)$ is the belonging case of e;

2) $\#_{\text{act}}(e)$ is the activity name of e;

3) $\#_{\text{trans}}(e)$ is the transaction type of e;

4) $\#_{\text{time}}(e)$ is the timestamp of e; and

5) $\#_{\text{vm}}(e)$ is the virtual machine information of e.

Definition 5: A case over some event universe ξ is a finite sequence of events $\sigma \in \xi^*$ such that each event appears only once and all events have the same case id, i.e., $1 \le i < j \le |\sigma|$: $\sigma(i) \ne \sigma(j) \wedge \#_{\text{case}}(\sigma(i)) = \#_{\text{case}}(\sigma(j))$. An event log is defined as a finite set of cases, i.e., $L \subseteq \xi^*$.

Consider for example the event log in Table I. It contains one case that records all events generated during one run of the upgrading process. In total 36 events are included and they are fully ordered based on timestamps. For example, we have the following observation for e_{10}: 1) $\#_{\text{case}}(e_{10}) = 1$ means the case id of e_{10} is 1; 2) $\#_{\text{act}}(e_{10}) = DI$ means the activity name of e_{10} is DI; 3) $\#_{\text{trans}}(e_{10}) = start$ means it is the start event; 4) $\#_{\text{time}}(e_{10}) =$ 04-05-2014 09 : 56 means the start event occurs at 04-05-2014 09 : 56; and 5) $\#_{\text{vm}}(e_{10}) = VM1$ indicates the upgrading process is working on virtual machine 1.

V. Formal Modeling of Multi-Instance Processes

In this section, we first introduce an informal description of multi-instance business process models. Then, we introduce the multi-instance Petri net, based on which the execution semantics of multi-instance business process models are defined.

A. Multi-Instance Business Processes

A multi-instance process is composed of a set of sub-processes such that they form a tree-like invocation relations. To specify the invocation relations, we introduce the Petri net with nested transitions where nested transitions may trigger multi-instance sub-processes.

Definition 6: A Petri net with nested transitions is a 2-tuple $PN_N = (PN, \mathcal{N})$ where

1) $PN = (P, T, F, l)$ is a labeled Petri net; and

2) $\mathcal{N} : T \to \{A, N\}$ is a mapping function such that $\forall t \in T$, $\mathcal{N}(t) = A$ represents t is an atomic transition and $\mathcal{N}(t) = N$ represents t is a nested transition.

Given a PN_N, we denote by $T_a = \{t \in T | \mathcal{N}(t) = A\}$ the atomic transition set and $T_n = \{t \in T | \mathcal{N}(t) = N\}$ the nested transition set. Specially, we use the notation T_{a0} and T_{n0} for PN_{N0}. The former is used to represent atomic activities while the latter represents nested activities that may trigger multi-instance sub-processes. Fig. 3 shows an example of the Petri net with nested transitions where single-line rectangles represent atomic transitions (e.g., SRU) and double-line rectangle represent nested transitions (e.g., UPI).

Fig. 3. An example of Petri net with nested transitions.

The definition of multi-instance process models is given.

Definition 7: A multi-instance business process model is defined as $MBPM = (Q, PN_{N0}, T^N, map)$ such that

1) Q is a set of PN_N;

2) PN_{N0} is the top-level process;

3) $T^N = \bigcup_{PN_N \in Q} T_n$ is the set of nested transitions; and

4) $map : T^N \to Q \setminus \{PN_{N0}\}$ is a function that maps each nested transition onto a sub-process that is instantiated multiple times.

An example of multi-instance business process is shown in Fig. 4. The top-level PN_N contains five normal transitions and one nested transition that refers to a Petri net. More specifically, nested transition UPI refers to PN_{N1}. Because PN_{N1} does not contain any nested transitions, the recursive definition stops at this level.

B. Multi-Instance Petri Nets

In the following, we introduce the semantic interpretation of

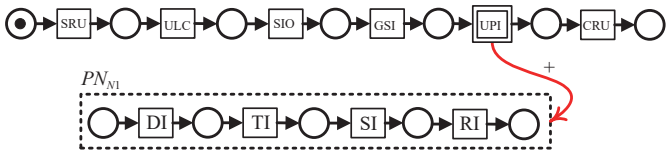

Fig. 4. A multi-instance business process model example.

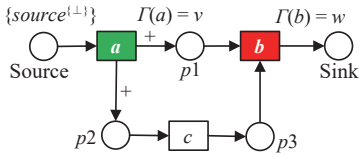

Fig. 5. An MPN example.

the multi-instance process model using multi-instance Petri net that is a type of extended Petri nets with distinguishable tokens as defined in the following sub-section.

Tokens of classical Petri nets (or workflow nets) are undistinguishable, therefore they cannot represent a process with multiple different instances explicitly. Based on the Petri net, we define our multi-instance Petri net (MPN) with the following extensions: 1) tokens are distinguishable for MPN, i.e., each token has a unique identifier; and 2) tokens can be produced (like ν-net [31]) and consumed during the net execution and each newly produced token explicitly refers to its parent token. The definition of MPN is given as follows.

Definition 8: A multi-instance Petri net is defined as a 6-tuple $MPN=(P,T,F,l,W,\Gamma)$, satisfying:

1) (P,T,F,l) is a Petri net;

2) $W: F\rightarrow\{1,+\}$ is a cardinality function for arcs;

3) $\Gamma: T\rightarrow\{\nu,\omega,\epsilon\}$ is a type function for transitions;

4) $\forall t\in T$, if $\Gamma(t)=\nu$ then $\forall p\in{}^\bullet t: W((p,t))=1$ and $\forall p_1, p_2\in t^\bullet: W((t,p_1))=W((t,p_2))$; and

5) $\forall t\in T$, if $\Gamma(t)\in\{\omega,\epsilon\}$ then $\forall p_1\in{}^\bullet t: W((p_1,t))=1$ and $\forall p_2\in t^\bullet: W((t,p_2))=1$.

Three types of transitions, including ν-transition, ω-transition, and ϵ-transition, are involved in Definition 8. The ν-transition is used to produce new tokens that refer back to an existing parent token and the ω-transition is used to consume a set of existing tokens and generates their parent token. Differently, the ϵ-transition is the same to traditional Petri net transitions as defined in Definition 2.

Before describing the formal execution semantics of an MPN, we first introduce some useful notations. Whenever we define MPN, we assume T^ϵ, T^ν, T^ω to be defined as follows: $T^\epsilon=\{t\in T|\Gamma(t)=\epsilon\}$ is the normal transition set, $T^\nu=\{t\in T|\Gamma(t)=\nu\}$ is ν-transition set, and $T^\omega=\{t\in T|\Gamma(t)=\omega\}$ is ω-transition set such that $T^\epsilon\cup T^\nu\cup T^\omega=T$, $T^\epsilon\cap T^\nu=\emptyset$, $T^\nu\cap T^\omega=\emptyset$ and $T^\epsilon\cap T^\omega=\emptyset$.

Fig. 5 shows an example of MPN and it can be formalized as follows according to Definition 8: $P=\{source,p1,p2,p3,sink\}, T=\{a,b,c\}, F=\{(source,a),(a,p1),(a,p2),(p1,b),(p2,c),(c,p3),(p3,b),(b,sink)\}, W=\{W((source,a))=1, W((a,p1))=+, W((a,p2))=+, W((p1,b)0=1, W((p2,c))=1, W((c,p3))=1, W((p3,b))=1, W((b,sink))=1\}$, and $\Gamma=\{\Gamma(a)=\nu,\Gamma(b)=\omega,\Gamma(c)=\epsilon\}$. Graphically, a ν-transition is represented by a rectangle colored with green, an ω-transition is represented by a rectangle colored with red and an ϵ-transition is represented by a rectangle colored with white.

After defining the syntax structure and basic notation of MPN, we define its execution semantics.

Let $\mathcal{I}$ be the universe of token identifiers. $\kappa: \mathcal{I}\rightarrow\mathcal{P}(\mathcal{I})$ is a function from parent identifiers to children identifiers, such that $i_1,i_2\in\mathcal{I}: \kappa(i_1)\cap\kappa(i_2)=\emptyset$. The token identifiers and their parent-children relationships form a tree structure, named token identifier tree. Note that a tree structure is essentially a directed acyclic graph. Formally, it is a 2-tuple $Tree=(\mathcal{I},E)$ where 1) $\mathcal{I}$ is the set of nodes; and 2) $E=\{(i_1,i_2)\in\mathcal{I}\times\mathcal{I}|i_2\in\kappa(i_1)\}$ is the set of edges. Specially, $\perp\in\mathcal{I}$ is the root token identifier of a tree and it does not have parent, i.e., $\nexists i_1\in\mathcal{I}: \perp\in\kappa(i_1)$.

A path in the tree is a sequence of nodes $i_1,i_2,\ldots,i_n$ such that $i_{n+1}\in\kappa(i_n)$ where $n\in\mathbb{N}$. The path from i_1 to i_n is denoted as $i_1\rightarrow i_n$. Specially, for any $i_1\in\mathcal{I}, i_1\neq\perp: \perp\rightarrow i_1$, i.e., there is always a path from the root to i_1.

Fig. 6 shows an example of the token identifier tree and it can be formalized as follows: $\mathcal{I}=\{\perp,m,n\}$, $\kappa(\perp)=\{m,n\}$ and $\kappa(m)=\kappa(n)=\emptyset$.

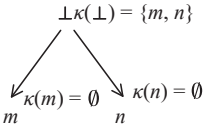

Fig. 6. A tree example.

Definition 9: A marking of MPN is a function $m: P\rightarrow\mathcal{P}(\mathcal{I})$ from places to token identifiers. The set of all markings is denoted as M.

For the example of MPN in Fig. 5, its initial marking is m_0 such that $m_0(source)=\{\perp\}$. In the following, we simply denote $m_0=\{source^{\{\perp\}}\}$.

Definition 10: A transition $t\in T$ is enabled at marking $m\in M$ for $i\in\mathcal{I}$ if and only if:

1) $\forall p\in{}^\bullet t: i\in m(p)\wedge\forall q\in t^\bullet, i\notin m(q)$ if $t\in T\setminus T^\omega$; and

2) $\forall p\in{}^\bullet t: \kappa(i)\subseteq m(p)\wedge\forall q\in t^\bullet, \kappa(i)\cap m(q)=\emptyset$ if $t\in T^\omega$.

Note that the enabledness of transitions in MPN is the same to that of an elementary net [32], i.e., each place contains each identifier at most once. According to Definition 10, transition a in Fig. 5 is enabled at m_0. An enabled transition t can fire, thereby changing the marking m to a new marking $m'\in M$. If an enabled transition fires, it consumes tokens from each of its input places and produces tokens to each of its output places. The transition firing rules are explained in the following definition.

Definition 11: Given a transition $t\in T$, it is enabled at marking $m\in M$ for $i\in\mathcal{I}$. The firing of t yields a new marking $m'\in M$, satisfying:

1) for any place $p\in{}^\bullet t$,

a) if $t\in T\setminus T^\omega$, $m'(p)=m(p)\setminus\{i\}$; and

b) if $t\in T^\omega$, $m'(p)=m(p)\setminus\kappa(i)$.

2) for any place $p\in t^\bullet$,

a) if $t\in T\setminus T^\nu$, $m'(p)=m(p)\cup\{i\}$; and

b) if $t\in T^\nu$, $m'(p)=m(p)\cup\kappa(i)$.

Given an MPN and its initial marking, we want to know

more about its behavior with respect to the firing rules as defined in Definition 11. Consider the *MPN* example in Fig. 5, the token identifier relationships in Fig. 6, and its initial marking $m_0 = \{source^{\{\perp\}}\}$. Transition a is enabled at m_0 and the firing of a will consume token $\perp$ from source and produce two tokens m and n to both $p1$ and $p2$. As shown in Fig. 7, the marking after firing transition a is denoted as $m_1 = \{p1^{\{m,n\}}, p2^{\{m,n\}}\}$. Then, transition c is enabled and can be fired until all tokens in $p2$ are moved one by one to $p3$ reaching marking $m_4 = \{p1^{\{m,n\}}, p3^{\{m,n\}}\}$. After this, transition b is enabled. The firing of b will consume all tokens in $p1$ and $p3$, and produce their parent token $\perp$ to sink. As depicted in Fig. 7, $m_5 = \{sink^{\{\perp\}}\}$ is reached finally.

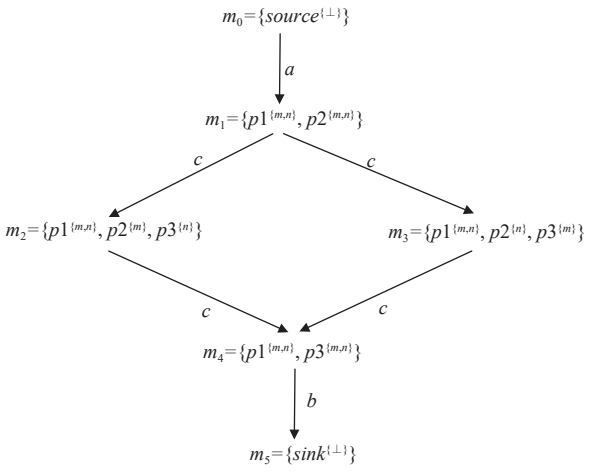

Fig. 7. One example reachability graph.

We define the semantic interpretation of MBPM based on the MPN constructs. Given an example of MBPM with t being the nested transition, and SP being the invoked multi-instance sub-process where p_s is the source place and p_c is the sink place. The transformation is shown in Fig. 8.

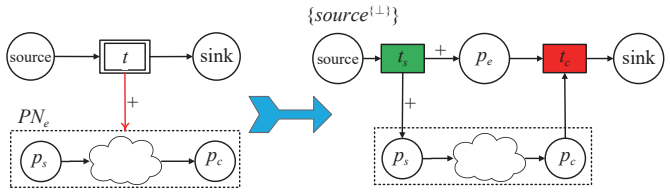

Fig. 8. MPN-based semantics.

Detailed transformation is defined as follows: 1) t is transformed to two transitions, denoted as t_s and t_c, and a place p_e is used to connect them with arcs (t_s, p_e) and (p_e, t_c); 2) arcs (t_s, p_s) and (p_c, t_c) are added to connect t_s and t_c as well as the invoked sub-process model; and 3) the following weight and type functions are given for transitions and arcs: $\Gamma(t_s) = \nu$, $W((t_s, p_e)) = +$, $W((t_s, p_s)) = +$, and $\Gamma(t_c) = \omega$.

VI. Discovering MBPM From Event Logs

This section introduces the discovery of MBPM from event logs with sub-process multi-instantiation information.

A. An Approach Overview

The whole MBPM discovery approach involves the following three steps:

Phase 1 (Nesting Relation Detection): By taking an event log as input, we first perform nesting relation detection among activities. The output is a nested activity relation tree where nesting relations among activities are represented as arcs.

Phase 2 (Hierarchical Event Log Construction): By taking an event log and the nested activity relation tree as input, we construct a hierarchical event log by using the nesting relations among activities. Note that we need to refactor each sub-log in such a way that no interleaved instances are included in the same trace.

Phase 3 (Multi-instance Business Process Model Discovery): After obtaining a hierarchical event log, we can visit different levels of logs and discover a process model for each sub-log. Note that any existing discovery approaches can be reused.

B. Nesting Relation Detection

Existing discovery algorithms cannot handle sub-processes or nesting relation among activities. To detect nesting relation, the next three basic ordering relations are introduced.

Definition 12: For any $a, b \in \mathcal{U}_A$ and $\sigma \in L$, we have

1) a is directly followed by b in σ, denoted as $a > b$, if there exists $i \in \{1, \ldots, |\sigma| - 1\}$: $\#_{act}(\sigma(i)) = a \wedge \#_{trans}(\sigma(i)) = complete \wedge \#_{act}(\sigma(i+1)) = b \wedge \#_{trans}(\sigma(i+1)) = start$;

2) a overlaps with b in σ, denoted as $a \gtrsim b$ if there exists $i, j, k, l \in \{1, \ldots, |\sigma|\}$ and $i < j < k < l$: $\#_{act}(\sigma(i)) = a \wedge \#_{trans}(\sigma(i)) = start \wedge \#_{act}(\sigma(j)) = b \wedge \#_{trans}(\sigma(j)) = start \wedge \#_{act}(\sigma(k)) = a \wedge \#_{trans}(\sigma(k)) = complete \wedge \#_{act}(\sigma(l)) = b \wedge \#_{trans}(\sigma(l)) = complete$; and

3) a contains b in σ, denoted as $a \triangleright b$ if there exists $i, j, k, l \in \{1, \ldots, |\sigma|\}$ and $i < j < k < l$: $\#_{act}(\sigma(i)) = a \wedge \#_{trans}(\sigma(i)) = start \wedge \#_{act}(\sigma(j)) = b \wedge \#_{trans}(\sigma(j)) = start \wedge \#_{act}(\sigma(k)) = b \wedge \#_{trans}(\sigma(k)) = complete \wedge \#_{act}(\sigma(l)) = a \wedge \#_{trans}(\sigma(l)) = complete$.

Definition 12 introduces the directly-follow, overlapping and containment. Then, nesting relations are defined as follows.

Definition 13: For any $a, b \in \mathcal{U}_A$, a and b are in nesting relation, denoted as $a \Subset b$, if and only if $(a \triangleright b) \wedge (b \not\triangleright a) \wedge (a \gtrsim b) \wedge (b \gtrsim a) \wedge (a \not> b) \wedge (b \not> a)$.

Similarity to casuality relation, concurrency relation, and choice relation, nesting relation is defined as a kind of derived relation among activities. The main difference is that the nesting relation is specifically for hierarchical process models while other derived relations are dedicated for flat process models. After formally introducing the nesting relation, process models with sub-processes can be discovered.

Definition 14: Let L be a lifecycle event log and $A \subseteq \mathcal{U}_A$ be its activity set. $NA(L) = \{a \in A | \exists b \in A: a \Subset b\}$ is the nesting activity set of L.

Consider the example log in Table I, its nesting activity set is $\{UPI\}$. After defining the nesting relations, we introduce a tree-like structure to represent the nesting relations among activities detected from a lifecycle event log.

Definition 15: The nesting relations among activities in A

form a tree structure, denoted as $(A, rootAct, \eta)$, such that:

1) $rootAct \subseteq A$ is a set of root activities such that

a) $\forall r_1, r_2 \in rootAct$: $r_1 \notin r_2 \wedge r_2 \notin r_1$, i.e., root activities have no nesting relation with each other; and

b) $\forall r \in rootAct, \nexists a \in A \setminus rootAct : a \Subset r$.

2) $\eta : A \nrightarrow \mathcal{P}(A) \setminus rootAct$ is a partial function that maps each activity onto its nesting activities.

a) for any $a \in A$, $b \in \eta(a)$: $a \Subset b$;

b) for any $b, c \in \eta(a)$: $b \notin c \wedge c \notin b$; and

c) for any $a, b \in A$: $\eta(a) \cap \eta(b) = \emptyset$, i.e., the nesting relations form a tree.

The activity nesting relation tree of the log in Table I is shown in Fig. 9, based on which we can see that: 1) SRU, ULC, SOI, GSI, UPI, and CRU are root activities; and 2) DI, TI, SI and RI are in nesting relation with UPI.

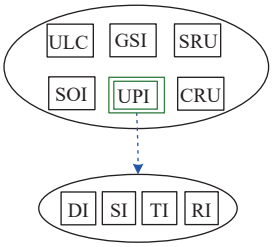

Fig. 9. An activity nesting relation tree example.

C. Hierarchical Log Construction and MBPM Discovery

After obtaining the activity nesting relation tree, a hierarchical event log can be constructed. Formal definition of hierarchical event log is given in the following.

Definition 16: Let L be an event log and $(A, rootAct, \eta)$ be the activity nesting relation tree of L. $(rootLog, \gamma)$ is the hierarchical event log of L where:

1) $rootLog = \bigcup_{\sigma \in L} \sigma \restriction_{\text{rootAct}}$ is the root event log of L; and

2) $\gamma : NA(L) \rightarrow \mathcal{U}_L$ is a function that maps a nesting activity to its sub-log. $\forall na \in NA(L)\}$, $\gamma(na) = \bigcup_{\sigma \in L} \sigma \restriction_{\eta(na)}$.

According to Definition 16, given an event log we can construct its hierarchical log by recursively referring to its nesting relation using the projection in Definition 1.

Before discovering MBPMs, we need to traverse different sub-logs to check if there exist multiple interleaved instances within the same traces. Consider for example the event log in Table I, the sub-log invoked by UPI contains one case that contains three interleaved sub-process instances. To guarantee the quality of discovered models, we need to refactor all sub-process logs in such a way that no interleaved instances are included in a single case. Please note that the sub-process instance identification is typically application-specific, and heavily relies on the specific domain knowledge of the underlying process. For our cloud resource management case, each sub-process instance is uniquely identified by the virtual machine information.

After refactoring all sub-logs, we are ready to discover MBPMs from hierarchical logs. We traverse different sub-logs and discover a process model for each sub-log. The inductive miner is used for sub-log discovery as it is the state-of-the-art algorithm [5]. Fig. 4 is the discovered MBPM by taking the event log in Table I as input.

VII. Tool Implementation

The proposed multi-instance business process model discovery approaches have been implemented as a plug-in (called multi-instance business process model discovery) in the open-source process mining framework ProM 6 [2]. It takes an XES-based event log as input, and returns an MBPM as output. A snapshot of this plug-in is shown in Fig. 10. In addition, to evaluate the quality of the discovered model with respect to the input event log, we have implemented another plugin, called, convert a multi-instance business process model to a flat Petri net. It takes an MBPM as input, and returns its corresponding flat Petri net. The following experiments are all based on these two plug-ins.

Fig. 10. Screenshot of the MBPM discovery plugin.

VIII. Experimental Evaluation

In this section, we perform a comparative evaluation using the event log collected from the cloud resource management case study in Section III. This event log contains 12 870 cases, and 360 360 events in total. Note that we have pre-processed the event log to be lifecycle consistent [9]. For these experiments, we used a laptop with a 2.40 GHz CPU, Windows 8.1 and Java SE 1.7.0 67 (64 bit) with 4 GB of allocated RAM.

A. Quality Metrics

To measure the quality of an MBPM against an event log, we need to first transform it to a flat Petri net. Therefore, existing quality metrics can be applied.

A transformation example is shown in Fig. 11, based on which the transformation rule is described as follows: 1) a nested transition t is transformed to two normal/flat transitions, denoted as t_s and t_c, and a place p_e is used to connect them with arcs (t_s, p_e) and (p_e, t_c); 2) arcs (t_s, p_s) and (p_c, t_c) are added to connect t_s and p_s, and p_c and t_c; and 3) a Petri net controller including transitions τ_1 and τ_2, place p_x, arcs (p_e, τ_1), (τ_1, p_e), (p_e, τ_2), (τ_2, p_e), (τ_1, p_x), (p_x, τ_2), (τ_1, p_s), (p_c, τ_2), is inserted to control the multiple instantiation behavior of the sub-process between p_s and p_c.

Fig. 12 shows the corresponding flat Petri net transformed

[2] https://svn.win.tue.nl/repos/prom/Packages/CongLiu

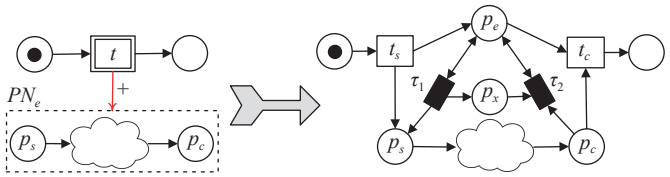

Fig. 11. Transformation from an MBPM to a flat Petri net.

from the MBPM in Fig. 4. Each nested transition in Fig. 4 is transformed to two normal transitions. For example, nested transition UPI is transformed to UPI_s and UPI_c. Specifically, UPI_s represents the start of activity UPI and UPI_c represents its completion. The sub-process is invoked after the start of UPI and completed before the completion of UPI.

Once we transform MBPMs to flat ones, we can use the well-defined metrics (fitness and precision) to measure the quality of the discovered models with respect to input logs.

a) Fitness: Fitness aims to measure the ability of the discovered model to replay the behavior recorded in the input event log. It measures the ability of the model to re-produce the behavior recorded in the event log. A fitness of 1 indicates that the discovered model can re-produce all traces of the input event log. Low fitness values normally indicate that the event log contains much more behaviour that is not allowed by the process model. Note that the fitness definition in [33] is adopted in this paper.

b) Precision: Precision aims to measure the ability of the discovered model to generate only the behavior recorded in the input event log. It measures the ability of the model to generate only the behavior recorded in the log. A precision of 1 indicates that all traces that are produced by the discovered model are contained in the input event log. Low precision values normally indicate that the process model contains much more behaviour that is not included in the event log. The precision definition in [34] is adopted in this paper.

According to [35], there is a trade-off between fitness and precision, adding a small fraction of behaviour in the event log may cause a slight decrease in fitness while the precision value may increase a lot. Therefore, we introduce the F-Measure.

c) F-measure: F-measure is defined as the harmonic mean of fitness and precision. Its definition is shown in the following equation:

$$F\text{-}measure = \frac{2 \times fitness \times precision}{fitness + precision}. \tag{2}$$

The F-measure only states how well the discovered model can reproduce the log. To quantify the understandability of discovered models, the complexity metric is introduced.

d) Complexity: The complexity of discovered models plays an important role in the comprehension. To quantify the complexity of discovered models, we use the following two complexity metrics with the assumption that models are represented by workflow nets [36]: i) extended Cardoso metric (ECaM); and ii) extended cyclomatic metric (ECyM). The ECaM metric measures the structural complexity of the model by counting the various splits (XOR, OR and AND) and gives each of them a certain penalty. The ECaM metric measures the behavioral complexity of the model by analyzing its reachability graph.

ECaM focuses on the syntax (structure) complexity of the model itself and ECyM focuses on the resulting behavior. The complexity metric is defined as the harmonic mean of ECaM and ECyM metrics. A low complexity value indicates that the component behavior is simple to understand.

$$Complexity = \frac{2 \times ECaM \times ECyM}{ECaM + ECyM}. \tag{3}$$

B. Experimental Results

To evaluate the effectiveness of the proposed approach, existing process discovery algorithms that can 1) handle lifecycle event logs and sub-processes; and 2) provide a publicly available implementation, are included. More specifically, we compare inductive miner lifecycle (IMlc) [37], statechart workbench (SW) [18] and hierarchical miner (HM) with noise threshold 0.85 [14] with our approach.

Detailed evaluation results are shown in Table II. Based on Table II, we have the following observations and explanations:

1) The F-measure values of our approach are much higher than those of other approaches. Normally, a high F-measure value indicates better model quality. It can be explained by the fact that our approach improves the precision of discovered models by identifying sub-processes and distinguishing interleaved sub-process instances.

2) Both HM and SW are able to identify nesting relations and sub-processes. However, when sub-processes are mixed with concurrent activities, SW cannot precisely distinguish concurrency and nesting relations. Therefore, HM performs better than SW. In addition, these two approaches cannot handle interleaved sub-process instances, and therefore, their F-measure values are lower than that of our approach.

3) As for the IMlc, it cannot handle sub-processes nor multiple instantiations, and therefore, they are simply treated as concurrency. As a result, the precision of discovered models is very low.

4) The complexity value of IMlc is the highest, i.e., IMlc leads to the most complex model. The complexity values of HM and SW are much lower than that of IMlc as these two approaches applied nesting relation detection to improve the quality of the discovered models. The complexity value of our approach is the lowest compared to the other three approaches, this is because interleaved sub-process instances are handled, and therefore, the understandability of the discovered model is improved.

As a conclusion, we show that the proposed approach can discover process models with better quality in terms of F-measure and understandability from logs containing lifecycle and multiple sub-process instantiation information compared with existing process discovery approaches.

IX. Conclusion

This paper proposes a novel process discovery technique to support multi-instance sub-process identification and MBPM discovery from event logs with sub-process multi-instantia-

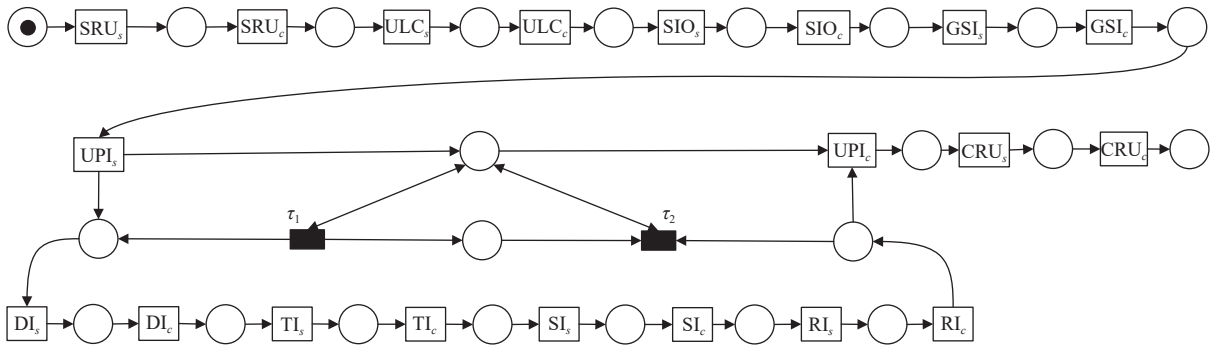

Fig. 12. Transformed flat Petri net from the MBPM in Fig. 4.

TABLE II
FITNESS, PRECISION AND F-MEASURE RESULTS

Approaches	Fitness	Precision	F-measure	Complexity
IMlc	0.87	0.33	0.48	26.5
SW	1	0.4	0.57	18.6
HM	0.84	0.72	0.77	16.2
Our approach	1	0.73	0.84	13.3

tion information. Formal semantics of discovered MBPM are precisely defined by MPNs. In addition, we also measure the quality of the discovered MBPMs against the input event logs by transforming an MBPM to a classical Petri net such that existing quality metrics, e.g., fitness and precision, can be used. All proposed approaches are fully implemented in the ProM toolkit. Using a cloud resource management case study, we compared our approach against the state-of-the-art process discovery techniques. The results demonstrate that our proposed approach outperforms existing approaches to discover process models with multi-instance sub-processes.

Our future work lies in quantitatively evaluating how the proposed nesting relation detection approach can handle noisy event logs. As for the sub-process instance identification, the current approach is application-specific and relies on the input event log. It is highly desired to propose general approaches by investigating temporal and interval relations among events [38]. In addition, we would like to evaluate the scalability of the proposed approach using large-scale real-life event logs.

REFERENCES

[1] W. M. P. Van der Aalst, *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. New York, USA: Springer, 2011.

[2] C. Liu, H. Duan, Q. T. Zeng, M. C. Zhou, F. M. Lu, and J. J. Cheng, "Towards comprehensive support for privacy preservation cross-organization business process mining," *IEEE Trans. Serv. Comput.*, vol. 12, no. 4, pp. 639–653, Jul.–Aug. 2019.

[3] W. van der Aalst, T. Weijters, and L. Maruster, "Workflow mining: Discovering process models from event logs," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 9, pp. 1128–1142, Sep. 2004.

[4] A. J. M. M. Weijters, W. M. P. van der Aalst, and A. K. A. De Medeiros, "Process mining with the HeuristicsMiner algorithm," Technische Universiteit Eindhoven, Einhofen, Netherlands, Tech. Rep. WP, 2006.

[5] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Discovering block-structured process models from event logs — A constructive approach," in *Proc. 34th Int. Conf. Applications and Theory of Petri Nets and Concurrency*, Milan, Italy, 2013, pp. 311–329.

[6] Q. T. Zeng, S. X. Sun, H. Duan, C. Liu, and H. Q. Wang, "Cross-organizational collaborative workflow mining from a multi-source log," *Decis. Support Syst.*, vol. 54, no. 3, pp. 1280–1301, Feb. 2013.

[7] I. Weber, M. Farshchi, J. Mendling, and J. G. Schneider, "Mining processes with multi-instantiation," in *Proc. 30th Annu. ACM Symp. Applied Computing*, Salamanca, Spain, 2015, pp. 1231–1237.

[8] L. J. Wen, J. M. Wang, W. M. P. van der Aalst, B. Q. Huang, and J. G. Sun, "A novel approach for process mining based on event types," *J. Intell. Inf. Syst.*, vol. 32, no. 2, pp. 163–190, Apr. 2009.

[9] L. J. Wen, W. M. P. van der Aalst, J. M. Wang, and J. G. Sun, "Mining process models with non-free-choice constructs," *Data Min. Knowl. Disc.*, vol. 15, no. 2, pp. 145–180, Oct. 2007.

[10] R. Conforti, M. Dumas, L. García-Bañuelos, and M. La Rosa, "BPMN miner: Automated discovery of BPMN process models with hierarchical structure," *Inf. Syst.*, vol. 56, pp. 284–303, Mar. 2016.

[11] X. X. Lu, M. Nagelkerke, D. van der Wiel, and D. Fahland, "Discovering interacting artifacts from ERP systems," *IEEE Trans. Serv. Comput.*, vol. 8, no. 6, pp. 861–873, Nov.–Dec. 2015.

[12] M. L. van Eck, N. Sidorova, and W. M. P. van der Aalst, "Multi-instance mining: Discovering synchronisation in artifact-centric processes," in *Proc. Int. Conf. Business Process Management*, Sydney, Australia, 2018, pp. 18–30.

[13] A. Kheldoun, K. Barkaoui, and M. Ioualalen, "Formal verification of complex business processes based on high-level petri nets," *Inf. Sci.*, vol. 385–386, pp. 39–54, Apr. 2017.

[14] C. Liu, "Hierarchical business process discovery: Identifying sub-processes using lifecycle information," in *Proc. Int. Conf. Web Services*, Beijing, China, 2020, pp. 423–427.

[15] C. Liu, L. Cheng, Q. T. Zeng, and L. J. Wen, "Formal modeling and discovery of hierarchical business processes: A Petri net based approach," *IEEE Trans. Syst., Man, and Cybern.: Syst.*, 2022. DOI: 10.1109/TSMC.2022.3195869

[16] C. Liu, "Automatic discovery of behavioral models from software execution data," *IEEE Trans. Autom. Sci. Eng.*, vol. 15, no. 4, pp. 1897–1908, Oct. 2018.

[17] C. Liu, "Discovery and quality evaluation of software component behavioral models," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 4, pp. 1538–1549, Oct. 2021.

[18] M. Leemans, W. M. P. van der Aalst, and M. G. J. van den Brand, "The statechart workbench: Enabling scalable software event log analysis using process mining," in *Proc. 25th Int. Conf. Software Analysis, Evolution and Reengineering*, Campobasso, Italy, 2018, pp. 502–506.

[19] M. H. Ghahramani, M. C. Zhou, and C. T. Hon, "Toward cloud computing QoS architecture: Analysis of cloud systems and cloud services," *IEEE/CAA J. Autom. Sinica*, vol. 4, no. 1, pp. 6–18, Jan. 2017.

[20] L. Wang, Y. Y. Du, and L. Qi, "Efficient deviation detection between a process model and event logs," *IEEE/CAA J. Autom. Sinica*, vol. 6, no. 6, pp. 1352–1364, Nov. 2019.

[21] Q. Mo, W. Song, F. Dai, L. L. Lin, and T. Li, "Development of collaborative business processes: A correctness enforcement approach," *IEEE Trans. Serv. Comput.*, vol. 15, no. 2, pp. 752–765, Mar.–Apr. 2022.

[22] C. Liu, Q. T. Zeng, L. Cheng, H. Duan, M. C. Zhou, and J. J. Cheng,

"Privacy-preserving behavioral correctness verification of cross-organizational workflow with task synchronization patterns," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 3, pp. 1037–1048, Jul. 2021.

[23] C. Liu, Q. T. Zeng, L. Cheng, H. Duan, and J. J. Cheng, "Measuring similarity for data-awarebusiness processes," *IEEE Trans. Autom. Science and Engineering*, vol. 19, no. 2, pp. 1070–1082, Apr. 2022.

[24] C. Liu, Y. L. Pei, L. Cheng, Q. T. Zeng, and H. Duan, "Sampling business process event logs using graph-based ranking model," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 5, pp. 1–14, 2021.

[25] C. Liu, H. L. Li, S. P. Zhang, L. Cheng, and Q. T. Zeng, "Cross-Department collaborative healthcare process model discovery from event logs", *IEEE Trans. Autom. Science and Engineering*, 2022. DOI: 10.1109/TASE.2022.3194312

[26] H. Duan, C. Liu, Q. T. Zeng, and M. C. Zhou, "Refinement-based hierarchical modeling and correctness verification of cross-organization collaborative emergency response processes," *IEEE Trans. Syst.*, *Man*, *Cybern.: Syst.*, vol. 50, no. 8, pp. 2845–2859, Aug. 2020.

[27] Q. T. Zeng, C. Liu, H. Duan, and M. C. Zhou, "Resource conflict checking and resolution controller design for cross-organization emergency response processes," *IEEE Trans. Syst.*, *Man*, *Cybern.: Syst.*, vol. 50, no. 10, pp. 3685–3700, Oct. 2020.

[28] S. G. Wang, W. L. Duo, X. Guo, X. N. Jiang, D. You, K. Barkaoui, and M. C. Zhou, "Computation of an emptiable minimal siphon in a subclass of Petri nets using mixed-integer programming," *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 1, pp. 219–226, Jan. 2021.

[29] W. Reisig, *Petri Nets: An Introduction, Vol 4*. Springer Science & Business Media, 2012.

[30] W. M. P. van der Aalst, "The application of petri nets to workflow management," *J. Circuits*, *Syst.*, *Comput.*, vol. 8, no. 1, pp. 21–66, Feb. 1998.

[31] F. Rosa-Velardo and D. de Frutos-Escrig, "Name creation vs. replication in petri net systems," *Fundam. Inform.*, vol. 88, no. 3, pp. 329–356, Aug. 2008.

[32] F. Rosa-Velardo and D. de Frutos-Escrig, "Decidability and complexity of petri nets with unordered data," *Theor. Comput. Sci.*, vol. 412, no. 34, pp. 4439–4451, Aug. 2011.

[33] A. Adriansyah, B. F. van Dongen, and W. M. P. van der Aalst, "Conformance checking using cost-based fitness analysis," in *Proc. 15th Int. Enterprise Distributed Object Computing Conf.*, Helsinki, Finland, 2011, pp. 55–64.

[34] A. Adriansyah, J. Munoz-Gama, J. Carmona, B. F. van Dongen, and W. M. P. van der Aalst, "Alignment based precision checking," in *Proc. Int. Conf. Business Process Management*, Tallinn, Estonia, 2012, pp. 137–149.

[35] J. De Weerdt, M. De Backer, J. Vanthienen, and B. Baesens, "A robust F-measure for evaluating discovered process models," in *Proc. IEEE Symp. Computational Intelligence and Data Mining*, Paris, France, 2011, pp. 148–155.

[36] K. B. Lassen and W. M. P. van der Aalst, "Complexity metrics for workflow nets," *Inf. Softw. Technol.*, vol. 51, no. 3, pp. 610–626, Mar. 2009.

[37] S. J. J. Leemans, D. Fahland, and W. M. P. van der Aalst, "Using life cycle information in process discovery," in *Proc. 13th Int. Conf. Business Process Management*, Innsbruck, Austria, 2016, pp. 204–217.

[38] J. G. Lou, Q. Fu, S. Q. Yang, J. Li, and B. Wu, "Mining program workflow from interleaved traces," in *Proc. 16th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, Washington, USA, 2010, pp. 613–622.

Cong Liu (Member, IEEE) received the B.S. and M.S. degrees in computer software and theory from Shandong University of Science and Technology, in 2013 and 2015, respectively, and the Ph.D. degree in computer science and information systems from the Eindhoven University of Technology, the Netherlands, in 2019. He is currently a Full Professor at the School of Computer Science and Technology, Shandong University of Technology. He has more than 60 publications in journals and conferences like *IEEE Transactions on Services Computing*, *IEEE Transactions on Systems*, *Man*, *and Cybernetics*: *Systems*, *IEEE Internet of Things Journal*, *IEEE Transactions on Automation Science and Engineering*, *IEEE Transactions on Big Data*, *IEEE Systems Journal*, *IEEE Transactions on Intelligent Transportation Systems*, *IEEE Network*, *Information Sciences*, *Future Generation Computer Systems*, *Expert Systems with Applications*, *Decision Support Systems*, *Enterprise Information Systems*, *International Conference on Web Services*, and *International Conference on Program Comprehension*, etc. His current research interests include business process management, process mining, Petri nets, and emergency management.