On Discovering Distributed Process Models – the case of asynchronous communication –

Pieter Kwantes and Jetty Kleijn

LIACS, Leiden University, P.O.Box 9512
NL-2300 RA Leiden, The Netherlands
{p.m.kwantes,h.c.m.kleijn}@liacs.leidenuniv.nl

Abstract. This paper is concerned with the discovery of models of distributed processes from event logs under the assumption that an algorithm for the discovery of the component models is given. We use Enterprise nets to model local processes, while Industry nets are compositions of Enterprise nets which interact through asynchronous message passing. First, we investigate the relation between the behaviour of Enterprise nets and that of Industry nets and formalise the (causal) structure of global (Industry net) behaviour in terms of a partial order derived from the message passing. Next, we demonstrate how to leverage existing algorithms for the discovery of workflow nets to discover Enterprise nets and how to combine these into an Industry net. Using the results on the structure of the global behaviour, we relate the behaviour of the Industry net thus discovered to the behaviour of the Enterprise nets and show how fitness of the Enterprise nets (the event log provided as input is included in the behaviour of the discovered net) is preserved as fitness of the Industry net.

Keywords: process discovery, distributed process, event log, workflow net, E-net, I-net, asynchronous channels, partial order

1 Introduction

Industry nets have been introduced in [18] as a framework to model global communication between enterprises where the design of their internal operations is left to the local level. In this set-up, operations at the enterprise level are represented by Enterprise nets (Petri nets with input, output, and internal transitions) and Industry nets are compositions of Enterprise nets that interact by exchanging messages through channels. In [18], a method is proposed to establish global compliance of an Industry net with a reference model by local checks (of Enterprise nets) only. In this paper, we focus on the discovery of Industry nets from a description of the (observed) global behaviour of a collection of collaborating enterprises.

Business process modelling and process mining are nowadays very active research areas and there are many approaches both to process discovery and conformance checking (see, eg., [6, 8, 11]). Here, we take advantage of this and consider the discovery of *distributed* processes (modelled as Industry nets) assuming the existence of an algorithm for the discovery of component processes (in the form of Enterprise nets with their own activities). To be precise, we first demonstrate how an algorithm that computes from an event log a Petri net, say a workflow net [7] (the Inductive Miner [19] is an example of such an algorithm), can be adapted to yield an Enterprise net. Then, given some global behaviour and the local event logs determined by it, an Industry net is constructed by combining the Enterprise nets discovered from the respective local event logs.

In order to compare the given global behaviour with the behaviour (the firing sequences) of the Industry net, we first investigate, after a preliminary section, in Sections 3, 4, and 5, the relation between the behaviour of an Industry net and the firing sequences of its Enterprise nets. As a consequence of the asynchronous set-up, every sequence that can be executed by an Industry net is also locally executable (ignoring the actions of other components) by each of its Enterprise nets. To describe the converse relationship, we formulate a prefix property that identifies those action sequences that can be executed by an Industry net whenever they are locally executable in all component Enterprise nets. Furthermore, by explicitly assigning occurrences of output actions to (dependent) occurrences of input actions, we can associate with each sequence with the prefix property, a labelled partial order reflecting the necessary ordering of action occurrences in this sequence. All linearisations of this labelled partial order are also firing sequences of the Industry net. We show that, in case the input/output relation is based on a FIFO relation, the set of these linearisations is maximal. In Section 6, we demonstrate how to leverage existing process discovery algorithms to the discovery of Industry nets. Then we argue, based on the results from the earlier sections, how fitness of the nets delivered by the original algorithm (the event log provided as input is included in the behaviour of the discovered net) guarantees the fitness of the distributed model. Section 7 concludes the paper and briefly considers related work.

Due to the page limit, this paper has no proofs. We provide, however, examples, explanations, and lemmas to clarify our statements and results.

2 Preliminaries

$\mathbb{N} = \{0, 1, 2, \ldots\}$ is the set of natural numbers including 0. For $n \in \mathbb{N}$, we set $[n] = \{1, 2, 3, \ldots, n\}$ and if $n = 0$, then $[n] = \emptyset$. By $A \uplus B$ we denote the

union of disjoint sets A and B. Given a partial order $R \subseteq A \times A$, we refer to a total order $R' \subseteq A \times A$ such that $R \subseteq R'$, as a *linearisation of* R.

An *alphabet* is a finite, non-empty, set of *symbols*. Let Σ be an alphabet. A *word* over Σ is a sequence $w = a_1 \cdots a_n$, with $n \geq 0$ and $a_i \in \Sigma$, for all $i \in [n]$; we refer to n as the *length* of w, denoted by $|w|$. If $n = 0$ then w is the *empty* word denoted by λ. Note that $|\lambda| = 0$. The set of all words over Σ is denoted as Σ^*. Any subset of Σ^* is a *language* (over Σ). It is often convenient to consider a word $w = a_1 \cdots a_n$ as a function $w : [|w|] \to \Sigma$, defined by $w(i) = a_i$ for all $i \in [n]$. The *alphabet* of w is $\mathtt{alph}(w) = \{w(i) \mid i \in [n]\}$. Hence $\mathtt{alph}(\lambda) = \emptyset$. For a language L, $\mathtt{Alph}(L) = \bigcup\{\mathtt{alph}(w) \mid w \in L\}$ is the set of all symbols that occur in the words of L. If a word w is a concatenation of words v_1 and v_2, i.e., $w = v_1 v_2$, then v_1 is said to be a *prefix* of w. The number of *occurrences of symbol* $a \in \Sigma$ *in a word* $w \in \Sigma^*$ is defined as $\#_a(w) = |\{i \mid w(i) = a\})|$ and *the set of occurrences of* w is $\mathtt{occ}(w) = \{(a, i) \mid a \in \mathtt{alph}(w) \wedge 1 \leq i \leq \#_a(w)\}$. In addition, we allocate a position with each occurrence of w through the function $\mathtt{pos}_w : \mathtt{occ}(w) \to [|w|]$ as follows: for all $(a, i) \in \mathtt{occ}(w)$, $\mathtt{pos}_w((a, i)) = k$ if $w(k) = a$ and $\#_a(w(1) \cdots w(k)) = i$. For a subset Δ of Σ, the projection of Σ^* on Δ^* is a function $\mathtt{proj}_{\Sigma,\Delta} : \Sigma^* \to \Delta^*$, defined by $\mathtt{proj}_{\Sigma,\Delta}(a) = a$ if $a \in \Delta$, and $\mathtt{proj}_{\Sigma,\Delta}(a) = \lambda$ otherwise; furthermore $\mathtt{proj}_{\Sigma,\Delta}(wa) = \mathtt{proj}_{\Sigma,\Delta}(w)\mathtt{proj}_{\Sigma,\Delta}(a)$ whenever $w \in \Sigma^*$ and $a \in \Sigma$. We omit the subscript Σ if it is clear from the context and thus write $\mathtt{proj}_{\Delta}(w)$ instead of $\mathtt{proj}_{\Sigma,\Delta}(w)$.

Petri nets A Petri net is a triple $N = (P, T, F)$, where P is a finite set of *places*, T is a finite non-empty set of *transitions* such that $P \cap T = \emptyset$, and $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs.
Let $N = (P, T, F)$ be a Petri net. If $N' = (P', T', F')$ is a Petri net such that $P \cup T$ and $P' \cup T'$ have no elements in common, then N and N' are *disjoint*. Let $x \in P \cup T$. Then ${}^\bullet x = \{y \mid (y, x) \in F\}$ and $x^\bullet = \{y \mid (x, y) \in F\}$ are the *preset* and the *postset*, respectively, of x (in N). A *marking* μ of N is a function $\mu : P \to \mathbb{N}$. Let $t \in T$ and μ a marking of N. Then t is *enabled at* μ if $\mu(p) > 0$ for all $p \in {}^\bullet t$. If t is enabled at μ, it may *occur*, and thus lead to a new marking μ' of N, denoted $\mu \xrightarrow{t}_N \mu'$, with $\mu'(p) = \mu(p) - 1$ whenever $p \in {}^\bullet t \setminus t^\bullet$; $\mu'(p) = \mu(p) + 1$ whenever $p \in t^\bullet \setminus {}^\bullet t$; and $\mu'(p) = \mu(p)$ otherwise. We extend the notation $\mu \xrightarrow{t}_N \mu'$ to sequences $w \in T^*$ as follows[1]: $\mu \xrightarrow{\lambda}_N \mu$ for all μ; and $\mu \xrightarrow{wt}_N \mu'$ for markings μ, μ' of N, $w \in T^*$ and $t \in T$, whenever there is a marking μ'' such that $\mu \xrightarrow{w}_N \mu''$ and $\mu'' \xrightarrow{t}_N \mu'$. If $\mu \xrightarrow{w}_N \mu'$, for some $w \in T^*$, we say that μ' is *reachable from* μ (in N). The

[1] We thus view T as an alphabet of action symbols.

language $\mathcal{L}(N,\mu) = \{w \in T^* \mid \mu \xrightarrow{w}_N \mu' \text{ for some marking } \mu'\}$ consists of all possible *firing sequences of N from μ*. A place $p \in P$ is a *source place* of N if ${}^\bullet p = \emptyset$ and it is a *sink place* of N if $p^\bullet = \emptyset$. The marking μ of N such that, for all $p \in P$, $\mu(p) = 1$ if p is a source place and $\mu(p) = 0$ otherwise, is the *default initial marking* of N. If μ is the default initial marking of N, we may write $\mathcal{L}(N)$ to denote $\mathcal{L}(N,\mu)$ and refer to it as the *language of N*. A *workflow net* (see [7]) is a Petri net with exactly one source and one sink place such that every place and every transition are on a path from source to sink.

Enterprise nets and Industry nets We recall the definitions of Enterprise and Industry nets from [18]. Enterprise nets (or E-nets, for short) are Petri nets equipped for asynchronous communication with other E-nets; they have transitions for inputting and transitions for outputting. An interaction between E-nets is realised by an occurrence of an output transition of one E-net and a (later) occurrence of an input transition of another E-net with a matching *message type*.
We let $\mathcal{M}$ denote the set of message types fixed throughout this paper.

Definition 1. *An Enterprise net is a tuple $\mathcal{E} = (P, [T_{int}, T_{inp}, T_{out}], F, M)$ such that T_{int}, T_{inp}, and T_{out} are mutually disjoint sets; T_{int} is the set of* internal transitions *of $\mathcal{E}$, T_{inp} its set of* input transitions*, and T_{out} its set of* output transitions*; furthermore, the* underlying Petri net of $\mathcal{E}$*, $und(\mathcal{E}) = (P, T_{int} \uplus T_{inp} \uplus T_{out}, F)$, is a Petri net with exactly one source place; finally $M : T_{inp} \cup T_{out} \to \mathcal{M}$ is the* communication function *of $\mathcal{E}$.* □

Given an Enterprise net $\mathcal{E}$, $\mathcal{L}(\mathcal{E}) = \mathcal{L}(und(\mathcal{E}))$ is *the language of $\mathcal{E}$*.
Note that E-nets, like workflow nets, have a single source place; no further restrictions are imposed on the underlying net.

Composing E-nets yields an Industry-net (or I-net, for short) with multiple source places. The E-nets involved are pairwise disjoint, i.e., their underlying Petri nets are disjoint. When combining them into an I-net, output and input transitions are matched via their message types.

Definition 2. *Let $n \geq 2$. Let $V = \{\mathcal{E}_i \mid i \in [n]\}$ be a set of pairwise disjoint E-nets with $\mathcal{E}_i = (P_i, [T_{i,int}, T_{i,inp}, T_{i,out}], F_i, M_i)$ for each $i \in [n]$.*
A matching over V *is a bijection $\varphi : \bigcup_{i\in[n]} T_{i,out} \to \bigcup_{j\in[n]} T_{j,inp}$ such that whenever $t \in T_{i,out}$ and $\varphi(t) \in T_{j,inp}$, for some i, j, then $i \neq j$ and $M_i(t) = M_j(\varphi(t))$.* □

A set V of mutually disjoint E-nets is said to be *composable* if there exists a matching over V. To construct an I-net from a composable set V and a matching φ over V, matching output and input transitions of the E-nets in V are connected through (new) channel places using channel arcs.

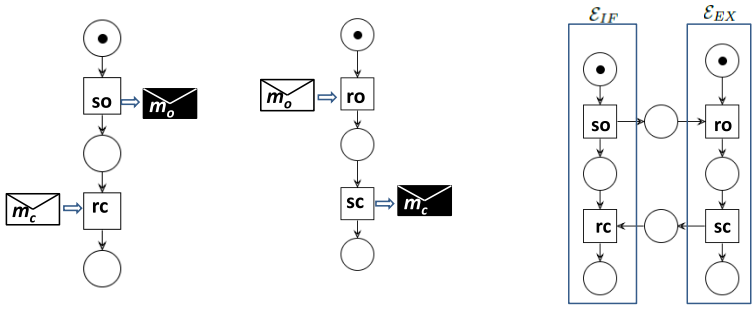
Definition 3. *Let $n \geq 2$. Let $V = \{\mathcal{E}_i : i \in [n]\}$ be a composable set of E-nets with $\mathcal{E}_i = (P_i, [T_{i,int}, T_{i,inp}, T_{i,out}], F_i, M_i)$ and $T_i = T_{i,int} \cup T_{i,inp} \cup T_{i,out}$ for all $i \in [n]$. Let φ be a matching over V.*
Then $P(V,\varphi) = \{[t, \varphi(t)] \mid t \in T_{i,out}, i \in [n]\}$ is the set of channel places *of V and φ, and $F(V,\varphi) = \{(t, [t, \varphi(t)]) \mid t \in T_{i,out}, i \in [n]\} \cup \{([t, \varphi(t)], \varphi(t)) \mid t \in T_{i,out}, i \in [n]\}$ is the set of* channel arcs *of V and φ. The sets $P(V,\varphi)$, $F(V,\varphi)$, and P_i, T_i, F_i, where $i \in [n]$, are all pairwise disjoint.*
The Industry net over (V,φ) *is the Petri net $\mathcal{I}(V,\varphi) = (P,T,F)$ with $P = \bigcup_{i\in[n]} P_i \cup P(V,\varphi)$, $T = \bigcup_{i\in[n]} T_i$, and $F = \bigcup_{i\in[n]} F_i \cup F(V,\varphi)$.* □

The notion of an E-net can be considered a generalisation of the notion of a workflow net through its underlying net. This generalisation is motivated, in particular, by the observation that workflows in enterprises often interact with workflows in other enterprises by a bilateral, asynchronous exchange of messages of a specified type (see eg., [20, 16, 24, 12]). The similarity between E-nets and workflow nets, moreover, allows us to leverage the large amount of research into the automated discovery of workflow nets, for the purpose of automated discovery of E-nets. The term "Industry", in "Industry net", is used loosely to refer to any group of collaborating enterprises.

Example 1. Consider two enterprises, an Investment Firm (*IF*) and an Exchange (*EX*), modelled by the E-nets $\mathcal{E}_{IF}$ and $\mathcal{E}_{EX}$ in Figs. 1a and 1b, respectively. The Investment Firm sends order messages to the Exchange (modelled by output transition **so** with message type m_o). The Exchange, upon receiving a message (via input transition **ro** with message type m_o), subsequently sends an order confirmation message to the Investment Firm (output transition **sc** with message type m_c). Then $\mathcal{I}(V,\varphi)$, the I-net in Fig.1c, with $V = \{\mathcal{E}_{IF}, \mathcal{E}_{EX}\}$ and φ defined by $\varphi(\mathbf{so}) = \mathbf{ro}$ and $\varphi(\mathbf{sc}) = \mathbf{rc}$, models the collaboration between the Investment Firm and the Exchange. Note that V allows only one matching.

□

The ideas underlying the concepts of E-nets and I-nets belong to a well-established branch of research concerned with composing systems modelled as Petri nets, into larger correctly functioning (concurrent) systems (see eg., [13, 14, 22, 23]). In [1] and also in [25] compositions of so-called open workflow nets are considered. These are Petri nets with interface places to communicate with other Petri nets. In [14], similar to the set-up of I-nets, local Petri nets do not have interface places but they communicate via distinguished input and output labels. As mentioned in [14], the advantage of this approach is that it leads to a better separation of concerns: eg., the designer of a local system does not have to consider how it will be used; this is a concern for the designer of the global system.



(a) E-net $\mathcal{E}_{IF}$ (b) E-net $\mathcal{E}_{EX}$ (c) The I-net

Fig. 1

3 E-net languages and I-net languages

Let V, φ, and $\mathcal{I}(V,\varphi) = (P,T,F)$ as specified in Definition 3, be fixed for this section, and for Section 4 and Section 5.

Clearly, the construction of $\mathcal{I}(V,\varphi)$ does not affect the internal structure of the E-nets in V and the set of source places of $\mathcal{I}(V,\varphi)$ consists of all source places of the $\mathcal{E}_i$. Moreover, removing channel places does not restrict the behaviour of an E-net. Consequently, all firing sequences in $\mathcal{L}(\mathcal{I}(V,\varphi))$ are combinations of firing sequences from the languages of the component nets.

Lemma 1. *If $w \in \mathcal{L}(\mathcal{I}(V,\varphi))$, then $\mathtt{proj}_{T_i}(w) \in \mathcal{L}(\mathcal{E}_i)$ for all $i \in [n]$.* □

On the other hand, the composition of E-nets into an I-net adds a channel place to the preset of each input transition. Consequently, the behaviour of an E-net may become restricted in the I-net. The property defined next describes how the occurrence of input transitions depends on the occurrence of output transitions.

Definition 4. *Let $w \in T^*$. Then w has the* prefix property *with respect to φ if $\#_a(u) \leq \#_{\varphi^{-1}(a)}(u)$ for all prefixes u of w and all $a \in T_{inp}$.*
A language $L \subseteq T^*$ has the prefix property with respect to φ if all $w \in L$ have this property. □

We will omit the reference to φ as it is fixed.

The asynchronous communication between its component E-nets guarantees that the firing sequences of $\mathcal{I}(V,\varphi)$ have the prefix property.

Lemma 2. *If $w \in \mathcal{L}(\mathcal{I}(V,\varphi))$, then w has the prefix property.* □

Conversely, any sequence $w \in T^*$ that can be (locally) executed by all E-nets and that satisfies the prefix property, belongs to $\mathcal{L}(\mathcal{I}(V,\varphi))$.

Lemma 3. *Let $w \in T^*$ be such that $\mathtt{proj}_{T_i}(w) \in \mathcal{L}(\mathcal{E}_i)$ for all $i \in [n]$. If w has the prefix property, then $w \in \mathcal{L}(\mathcal{I}(V,\varphi))$.* □

Example 2. (Ex. 1 ctd.) Let $T = \{\mathbf{so}, \mathbf{rc}, \mathbf{ro}, \mathbf{sc}\}$. Let $w = \langle \mathbf{so}, \mathbf{ro}, \mathbf{sc}, \mathbf{rc}\rangle$.[2] Now $\mathtt{proj}_{T_{IF}}(w) = \langle \mathbf{so}, \mathbf{rc}\rangle \in \mathcal{L}(\mathcal{E}_{IF})$ and $\mathtt{proj}_{T_{EX}}(w) = \langle \mathbf{ro}, \mathbf{sc}\rangle \in \mathcal{L}(\mathcal{E}_{EX})$, where T_{IF} and T_{EX} are the sets of transitions of $\mathcal{E}_{IF}$ and $\mathcal{E}_{EX}$, respectively. Obviously, w has the prefix property. Thus $w \in \mathcal{L}(\mathcal{I}(V,\varphi))$ by Lemma 3. For $w' = \langle \mathbf{so}, \mathbf{ro}, \mathbf{rc}, \mathbf{sc}\rangle$, the prefix property does not hold: in $u = \langle \mathbf{so}, \mathbf{ro}, \mathbf{rc}\rangle$, a prefix of w', we have $\#_{\mathbf{rc}}(u) = 1$, but $\#_{\mathbf{sc}}(u) = 0$. Indeed, $w' \notin \mathcal{L}(\mathcal{I}(V,\varphi))$ although $\mathtt{proj}_{T_{IF}}(w') = \mathtt{proj}_{T_{IF}}(w)$ and $\mathtt{proj}_{T_{EX}}(w') = \mathtt{proj}_{T_{EX}}(w)$. □

Combining the above three lemmas yields

Theorem 1. *Let $L \subseteq T^*$. Then $L \subseteq \mathcal{L}(\mathcal{I}(V,\varphi))$ if and only if L has the prefix property and $\mathtt{proj}_{T_i}(L) \subseteq \mathcal{L}(\mathcal{E}_i)$ for all $i \in [n]$.* □

From Lemma 1 and Theorem 1, it follows that whenever there exists a language L with the prefix property and for which $\mathtt{proj}_{T_i}(L) = \mathcal{L}(\mathcal{E}_i)$ for all $i \in [n]$, the I-net captures the full unrestricted behaviour of each E-net.

Corollary 1. *Let $L \subseteq T^*$ be such that L has the prefix property and $\mathtt{proj}_{T_i}(L) = \mathcal{L}(\mathcal{E}_i)$ for all $i \in [n]$. Then $\mathtt{proj}_{T_i}(\mathcal{L}(\mathcal{I}(V,\varphi))) = \mathcal{L}(\mathcal{E}_i)$ for all $i \in [n]$.* □

It should however be noted, that the conditions on L in Corollary 1 do not imply that $\mathcal{L}(\mathcal{I}(V,\varphi)) = L$. By Theorem 1, $L \subseteq \mathcal{L}(\mathcal{I}(V,\varphi))$ and as we argue next, this inclusion may be strict.

4 Structuring words with the prefix property

Whereas the prefix property is based on a simple comparison of numbers of occurrences, assignment functions, as defined next, relate each occurrence of an input transition to a corresponding occurrence of an output transition.

Definition 5. *An* assignment function *with respect to φ for a word $w \in T^*$ is an injective function $\theta : (\mathtt{occ}(w) \cap (T_{inp} \times \mathbb{N})) \to \mathtt{occ}(w)$ such that for every occurrence $(b,j) \in \mathtt{occ}(w)$ with $b \in T_{inp}$, $\theta(b,j) = (a,i)$ where $a = \varphi^{-1}(b)$ and i is such that $\mathtt{pos}_w(a,i) < \mathtt{pos}_w(b,j)$.* □

[2] The elements of the alphabet T are symbols consisting of two letters. In order to avoid confusion, we denote in this and similar examples, any sequence $a_1 \cdots a_n$ with $a_i \in T$ for each $i \in [n]$ by $\langle a_1, \cdots, a_n\rangle$.

Again, since φ is fixed, we will omit the reference to φ.

Remark 1. Clearly, every word for which an assignment function exists, has the prefix property. Conversely, whenever a word has the prefix property it has at least one (in general, more than one) assignment function. □

Example 3. Assume $T = \{a, b\}$ and $\varphi(a) = b$. Let $w = aabb$. Then w has two assignment functions: θ defined by $\theta(b,1) = (a,1)$ and $\theta(b,2) = (a,2)$; and θ' defined by $\theta'(b,1) = (a,2)$ and $\theta'(b,2) = (a,1)$. □

Each assignment function for w determines a partial order on $\mathtt{occ}(w)$.

Definition 6. *Let $w \in T^*$ and let θ be an assignment function for w. Let $(a,i),(b,j) \in \mathtt{occ}(w)$, with $a \in T_k$ and $b \in T_l$ for some $k, l \in [n]$.*
Then $(a,i) \leq_{w,\theta} (b,j)$ if
(1) *$k = l$ and $\mathtt{pos}_w(a,i) \leq \mathtt{pos}_w(b,j)$ or*
(2) *$k \neq l$ and $b \in T_{inp}$ and $\theta(b,j) = (a,i)$.* □

By condition (1), $\leq_{w,\theta}$ incorporates the ordering of occurrences of transitions in w that originate from the same E-net; by (2), it reflects the order of occurrences of input transitions and their assigned occurrences of output transitions.

Lemma 4. *Let $w \in T^*$ and let θ be an assignment function for w. Then $\leq^+_{w,\theta}$, the transitive closure of $\leq_{w,\theta}$, is a partial order on $\mathtt{occ}(w)$.* □

The linearisations of $\leq^+_{w,\theta}$, where w is a word and θ an assignment function for w, define a set of words that can be obtained from w by (repeatedly) interchanging the positions of unrelated occurrences.

Definition 7. *Let $w \in T^*$ and let θ be an assignment function for w. Then $\mathtt{lin}_\theta(w) = \{v \in T^* \mid \mathtt{occ}(v) = \mathtt{occ}(w)$ and for all $x, y \in \mathtt{occ}(v), x \leq^+_{w,\theta} y$ implies $\mathtt{pos}_v(x) \leq \mathtt{pos}_v(y)\}$ is the set of* θ-linearisations of w. □

Clearly $w \in \mathtt{lin}_\theta(w)$ whenever $\mathtt{lin}_\theta(w)$ is defined.

Remark 2. Let w and θ as in Definition 7 and let $v \in \mathtt{lin}_\theta(w)$. Then
(1) $\mathtt{proj}_{T_i}(v) = \mathtt{proj}_{T_i}(w)$ for all $i \in [n]$ and
(2) θ is an assignment function for v and $\mathtt{lin}_\theta(v) = \mathtt{lin}_\theta(w)$.
From (2) and Remark 1, it follows that v has the prefix property. □

Example 4. (Ex. 3 ctd.) Let $v = abab$. Then $\mathtt{occ}(v) = \mathtt{occ}(w)$ and $\mathtt{lin}_\theta(w) = \{v, w\}$. Clearly, θ is an assignment function for v and $\mathtt{lin}_\theta(v) = \mathtt{lin}_\theta(w)$. Note that $\mathtt{lin}_{\theta'}(w) = \{w\}$ and θ' is not an assignment function for v. □

By combining Remark 2 with Lemma 3, the statement of Lemma 3 can be strengthened. This shows that whenever a word with the prefix property can be locally executed by all component E-nets, then, for all its assignment functions θ, each of its θ-linearisations can be executed globally by the I-net.

Theorem 2. *Let $w \in T^*$ be such that $\mathtt{proj}_{T_i}(w) \in \mathcal{L}(\mathcal{E}_i)$ for all $i \in [n]$. If θ is an assignment function for w, then $\mathtt{lin}_\theta(w) \subseteq \mathcal{L}(\mathcal{I}(V, \varphi))$.* □

Example 5. The I-net in Fig. 2 is an extension of the I-net from Example 2. The Exchange, after sending a confirmation (output transition **sc**) to the Investment Firm, can send (output transition **si**) a settlement instruction to the *Central Securities Depositary* (*CD*) to transfer ordered securities to the Investment Firm. The Central Securities Depositary - after receiving (input transition **ri** matching **si**) the settlement instruction - sends (output transition **ss**) a confirmation of settlement to the Investment Firm (that receives the message via input transition **rs**).

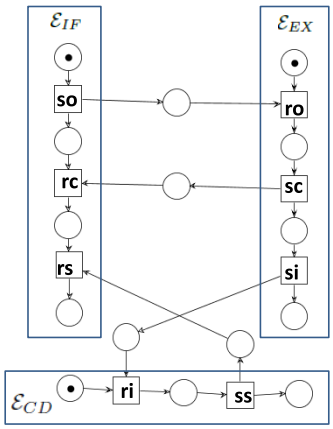


Fig. 2: I-net composed of $\mathcal{E}_{IF}$, $\mathcal{E}_{EX}$ and $\mathcal{E}_{CD}$

Let $w = \langle \mathbf{so}, \mathbf{ro}, \mathbf{sc}, \mathbf{rc}, \mathbf{si}, \mathbf{ri}, \mathbf{ss}, \mathbf{rs} \rangle$. So, w can be (locally) executed by each of the three enterprise nets in Fig. 2. Also, w has the prefix property (w.r.t. φ displayed in Fig. 2). Hence w is in the language of the I-net in Fig. 2. Define θ by $\theta(\mathbf{ro}, 1) = (\mathbf{so}, 1)$, $\theta(\mathbf{rc}, 1) = (\mathbf{sc}, 1)$, $\theta(\mathbf{ri}, 1) = (\mathbf{si}, 1)$, and $\theta(\mathbf{rs}, 1) = (\mathbf{ss}, 1)$. This θ is an (the only) assignment function for w (w.r.t. φ). Consider $v = \langle \mathbf{so}, \mathbf{ro}, \mathbf{sc}, \mathbf{si}, \mathbf{rc}, \mathbf{ri}, \mathbf{ss}, \mathbf{rs} \rangle$, obtained from w by exchanging $(\mathbf{si}, 1)$ and $(\mathbf{rc}, 1)$. These occurrences are not ordered by $\leq^+_{w,\theta}$. Hence $v \in \mathtt{lin}_\theta(w)$ and v is, like w, in the language of the I-net. □

5 Assignment functions of type FIFO

By Theorem 2, every assignment function θ for a word w with the prefix property determines a set of words $\mathtt{lin}_\theta(w)$ that can be executed by the I-net, provided w can be locally executed by all component E-nets. As remarked earlier, a word with the prefix property may have more than one assignment function. In this section we demonstrate that considering one particular type of assignment function is sufficient to describe all possible linearisations. Intuitively, in terms of the I-net, one could say that each assignment function describes for each occurrence of an input transition, which token it should take from its input channel place (namely the token deposited by the assigned occurrence of its output transition). Based on this point of view, the following two types of assignment functions represent natural policies to deal with the messages (tokens) in the channel places.

Definition 8. *Let $w \in T^*$ and let θ be an assignment function for w. Then*
(1) *θ is of* type FIFO *with respect to φ if, $\theta(b,j) = (\varphi^{-1}(b), j)$ for every $(b,j) \in \mathtt{occ}(w)$ such that $b \in T_{inp}$;*
(2) *θ is of* type Stack *with respect to φ if, for every $(b,j) \in \mathtt{occ}(w)$ such that $b \in T_{inp}$, $\theta(b,j) = (a,i)$ where $a = \varphi^{-1}(b)$, $w = xaybz$, for some $x,y,z \in occ(w)$ such that $\#_b(y) = \#_a(y)$, and $j = \#_b(xayb)$ and $i = \#_a(xa)$.* □

Again, since φ is fixed, we will omit the reference to φ.

Every word with the prefix property has one assignment function of type FIFO and one of type Stack (and these functions may be the same).

Example 6. (Ex. 3 and Ex. 4 ctd.) For w, the assignment function θ is of type FIFO and θ' is of type Stack. Moreover, θ is an assignment function for v of type FIFO and of type Stack. □

In the sequel, $\theta_w^{\mathtt{fifo}}$ denotes the assignment function of type FIFO of any word $w \in T^*$ with the prefix property. In addition, if $L \subseteq T^*$ is a language that has the prefix property, then $\mathtt{lin}_{\mathtt{FIFO}}(L) = \bigcup\{\mathtt{lin}_{\theta_w^{\mathtt{fifo}}}(w) \mid w \in L\}$ consists of all *FIFO-linearisations of L*.

Lemma 5. *Let $v, w \in T^*$ be such that v and w have the prefix property and $\mathtt{occ}(v) = \mathtt{occ}(w)$. Then $\theta_v^{\mathtt{fifo}} = \theta_w^{\mathtt{fifo}}$.* □

This observation does not hold for all assignment functions as can be seen in Example 6. Assignment functions of type FIFO are also special as they impose the least restrictive ordering (with for each occurrence of an input transition, only one occurrence of an output transition that precedes it). and thus a maximal (w.r.t. set inclusion) set of linearisations.

Lemma 6. *Let $w \in T^*$ have the prefix property. Then $\mathtt{lin}_{\theta^{\mathtt{fifo}}_w}(w) = \{v \in T^* \mid v$ has the prefix property and $\mathtt{proj}_{T_i}(v) = \mathtt{proj}_{T_i}(w)$ for all $i \in [n]\}$.* □

The inclusion from left to right is true for all assignment functions by Remark 2. The converse follows from Lemma 5 and Definition 7.

Example 7. Let $\mathcal{I}(V, \varphi)$ be the I-net in Fig. 3 (extending Fig.1c). The Investment Firm from Example 2 can now send multiple orders to the Exchange. This is modelled using internal transitions in the E-nets: **ii1** and **ii2** for the Investment Firm and **ei1** and **ei2** for the Exchange.

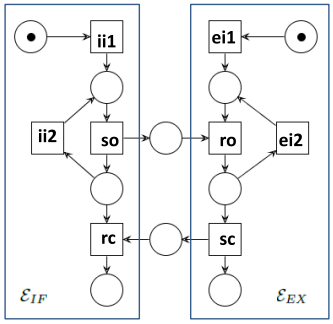


Fig. 3: I-net with option to repeat orders

Let $v = \langle \mathbf{ei1}, \mathbf{ii1}, \mathbf{so}, \mathbf{ii2}, \mathbf{ro}, \mathbf{ei2}, \mathbf{so}, \mathbf{ii2}, \mathbf{ro}, \mathbf{ei2}, \mathbf{so}, \mathbf{ro}, \mathbf{sc}, \mathbf{rc} \rangle$ and let $w = \langle \mathbf{ei1}, \mathbf{ii1}, \mathbf{so}, \mathbf{ii2}, \mathbf{so}, \mathbf{ii2}, \mathbf{so}, \mathbf{ro}, \mathbf{ei2}, \mathbf{ro}, \mathbf{ei2}, \mathbf{ro}, \mathbf{sc}, \mathbf{rc} \rangle$. Clearly, both v and w have the prefix property and $\mathtt{proj}_{T_i}(v) = \mathtt{proj}_{T_i}(w)$ for $i \in \{1, 2\}$ with $T_1 = \{\mathbf{ii1}, \mathbf{ii2}, \mathbf{so}, \mathbf{rc}\}$ and $T_2 = \{\mathbf{ei1}, \mathbf{ei2}, \mathbf{ro}, \mathbf{sc}\}$. Let θ be the assignment function for w of type FIFO. Hence, by Lemma 6, $v \in \mathtt{lin}_\theta(w)$. However, for the assignment function θ' of w of type Stack (and thus $\theta'(\mathbf{ro}, 1) = (\mathbf{so,3})$), $v \in \mathtt{lin}_{\theta'}(w)$ does not hold. □

For a language $L \subseteq T^*$ that has the prefix property, we denote by $\mathtt{lin}(L)$ the language consisting of all words that can be obtained as a θ-linearisation of any word $w \in L$ in for whatever assignment function θ for w. Thus, $\mathtt{lin}(L) = \bigcup\{\mathtt{lin}_\theta(w) \mid w \in L$ and θ an assignment function for $w\}$.

Using Lemma 6, we obtain the following extension of Theorem 2.

Theorem 3. *Let $L \subseteq T^*$ be a language with the prefix property such that $\mathtt{proj}_{T_i}(L) \subseteq \mathcal{L}(\mathcal{E}_i)$ for all $i \in [n]$.*
Then $L \subseteq \mathtt{lin}(L) \subseteq \mathtt{lin}_{\mathtt{FIFO}}(L) \subseteq \mathcal{L}(\mathcal{I}(V, \varphi))$. □

This theorem together with Theorem 1 shows that the languages of I-nets are closed under exchanging independent occurrences of transitions.

Corollary 2. $\mathtt{lin}(\mathcal{L}(\mathcal{I}(V,\varphi))) = \mathtt{lin}_{\mathtt{FIFO}}(\mathcal{L}(\mathcal{I}(V,\varphi))) = \mathcal{L}(\mathcal{I}(V,\varphi))$. □

6 Discovery of I-nets

We are now ready to address our initial question: given an event log (language), representing the observed behaviour of a given number of distinct, collaborating enterprises, construct an I-net that generates this observed behaviour. In this section, we will demonstrate how we can leverage existing algorithms for the discovery of isolated (local) processes, to solve this problem. A formal definition of such a process discovery algorithm, based on [6], is given next.

Definition 9. *Let $\mathbb{L}$ be a family of languages. A* process discovery algorithm *$\mathcal{A}$ for $\mathbb{L}$ is an algorithm that computes for all $L \in \mathbb{L}$, a Petri net $\mathcal{A}(L) = (P, T, F)$ with a single source place and such that $T = \mathtt{Alph}(L)$ and $L \subseteq \mathcal{L}(\mathcal{A}(L))$.* □

To leverage such algorithms to a system of asynchronously communicating nets, one needs to describe the distribution of actions over components in combination with their role as internal, input, or output action. This leads to the concept of a distributed communicating alphabet defined next.

Definition 10. *Let $n \geq 1$. An* n-dimensional distributed communicating alphabet *(or* n-DCA*, for short) is a tuple*
$\mathcal{DA} = ([\Sigma_1, \ldots, \Sigma_n], [\Sigma_{int}, \Sigma_{inp}, \Sigma_{out}], mt, cp)$ *such that*
– *$\Sigma_1, \ldots, \Sigma_n$ are non-empty, pairwise disjoint alphabets;*
– *Σ_{int}, Σ_{inp}, Σ_{out} are pairwise disjoint alphabets consisting of* internal actions*,* input actions *and* output actions*, respectively;*
– *$\bigcup_{i\in[n]} \Sigma_i = \Sigma_{int} \uplus \Sigma_{inp} \uplus \Sigma_{out}$;*
– *$mt : \Sigma_{inp} \cup \Sigma_{out} \to \mathcal{M}$ is a function that assigns message types to the input and output actions;*
– *$cp : \Sigma_{inp} \cup \Sigma_{out} \to \Sigma_{inp} \cup \Sigma_{out}$ is a (*complementing*) bijection, which is not defined ($cp = \emptyset$) if $n = 1$, and otherwise ($n \geq 2$), for all $a \in \Sigma_{inp} \cup \Sigma_{out}$: if $a \in \Sigma_{inp}$, then $cp(a) \in \Sigma_{out}$ and if $a \in \Sigma_{out}$, then $cp(a) \in \Sigma_{inp}$;*
if $a \in \Sigma_i$ for some $i \in [n]$, then $cp(a) \in \Sigma_j$ where $j \in [n]$ is such that $i \neq j$;
$mt(cp(a)) = mt(a)$; and $cp(cp(a)) = a$. □

The alphabet $\bigcup_{i\in[n]} \Sigma_i = \Sigma_{int} \uplus \Sigma_{inp} \uplus \Sigma_{out}$ in Definition 10, also referred to as the *underlying alphabet of the n-DCA $\mathcal{DA}$*, is intended to represent the actions available to n interacting enterprises. $\mathcal{DA}$ describes both their

distribution across the enterprises and their interaction capacities. A 1-*DCA* consists of a single enterprise. When specifying a 1-*DCA*, we can omit its complementing bijection *cp* as it is not defined (and not needed). For $i \in [n]$, we refer to the 1-*DCA* $\mathcal{DA}_i = ([\Sigma_i], [\Sigma_{i,int}, \Sigma_{i,inp}, \Sigma_{i,out}], mt_i)$ as the *i-th component of* $\mathcal{DA}$.

For the rest of this section we assume a fixed family of languages $\mathbb{L}$ and a fixed process discovery algorithm $\mathcal{A}$ for $\mathbb{L}$. The following definition describes how $\mathcal{A}$ can be used to discover E-nets.

Definition 11. *The* E-net discovery algorithm derived from $\mathcal{A}$*, is the algorithm* $\mathcal{A}_E$ *that computes, for all pairs* $(L, \mathcal{DA})$ *such that* $L \in \mathbb{L}$ *with* $\mathcal{A}(L) = (P, T, F)$*, and* $\mathcal{DA} = ([T], [T_{int}, T_{inp}, T_{out}], mt)$ *is a 1-*DCA *with* $\mathtt{Alph}(L) = T$ *as its underlying alphabet, the E-net* $\mathcal{A}_E(L, \mathcal{DA}) = (P, [T_{int}, T_{inp}, T_{out}], F, mt)$*.* □

Thus $\mathcal{A}_E$ adds the information from $\mathcal{DA}$ to the transitions of $\mathcal{A}(L)$. Clearly, $\mathcal{A}_E(L, \mathcal{DA})$ is an E-net. [3] From Definitions 9 and 11, we have $L \subseteq \mathcal{L}(\mathcal{A}(L)) = \mathcal{L}(\mathcal{E})$. Now we move to the discovery of I-nets on basis of $\mathcal{A}$.

Definition 12. *The* I-net discovery algorithm derived from $\mathcal{A}$*, is the algorithm* $\mathcal{A}_I$ *that computes, for all pairs* $(L, \mathcal{DA})$ *such that*
– $\mathcal{DA} = ([\Sigma_1, \ldots, \Sigma_n], [\Sigma_{int}, \Sigma_{inp}, \Sigma_{out}], mt, cp)$ *is an* n-DCA*for an* $n \geq 2$*,*
– $\mathtt{Alph}(L) = \bigcup_{i \in [n]} \Sigma_i$*, the underlying alphabet of* $\mathcal{DA}$*, and*
– $\mathtt{proj}_{\Sigma_i}(L) \in \mathbb{L}$ *for all* $i \in [n]$*,*
the I-net $\mathcal{A}_I(L, \mathcal{DA}) = \mathcal{I}(V, \varphi)$ *with* $V = \{\mathcal{A}_E(\mathtt{proj}_{\Sigma_i}(L), \mathcal{DA}_i) \mid i \in [n]\}$ *and* $\varphi(a) = cp(a)$ *for all* $a \in \Sigma_{out}$*.* □

Note that $\mathcal{A}_I(L, \mathcal{DA}) = \mathcal{I}(V, \varphi)$ in Definition 12 is indeed an I-net, since, by Definition 11, for all $i \in [n]$, $\mathcal{A}_E(\mathtt{proj}_{\Sigma_i}(L), \mathcal{DA}_i)$ is an E-net with set of transitions Σ_i; the Σ_i are mutually disjoint; and the properties of *cp* described in Definition 10 guarantee that φ is a matching for V.

By Definitions 9 and 12, we can transfer Theorem 1, Corollary 1, and Theorem 3 to the setting of discovering an I-net from a given language L.

Theorem 4. *Let* L *be a language with the prefix property. Let* $n \geq 2$ *and let* $\mathcal{DA}$ *be an* n*-DCA with* $\mathtt{Alph}(L)$ *as its underlying alphabet*
Let $\mathcal{A}_I(L, \mathcal{DA}) = \mathcal{I}(V, \varphi)$ *with* $V = \{\mathcal{E}_i \mid i \in [n]\}$*. Then*
(1) $L \subseteq \mathtt{lin}(L) \subseteq \mathtt{lin}_{\mathtt{FIFO}}(L) \subseteq \mathcal{L}(\mathcal{I}(V, \varphi))$*;*
(2) $\mathtt{proj}_{\Sigma_i}(\mathcal{L}(\mathcal{I}(V, \varphi))) = \mathtt{proj}_{\Sigma_i}(L)$ *if* $\mathtt{proj}_{\Sigma_i}(L) = \mathcal{L}(\mathcal{E}_i)$ *for all* $i \in [n]$*, where for each* $i \in [n]$*,* Σ_i *is the underlying alphabet of* $\mathcal{DA}_i$*.* □

[3] In case $\mathcal{A}$ is an algorithm for the discovery of workflow nets, like the Inductive Miner [19], $\mathcal{A}_E(L, \mathcal{DA})$ would have the structure of a workflow net.

Thus, given a language L with the prefix property representing an event log of a system of n enterprises, and an n-dimensional distributed communicating alphabet, we can construct an I-net as described in Definition 12. This I-net has all words in L as firing sequences together with all their linearisations (obtained by exchanging occurrences of independent actions). In fact, it is sufficient to consider only their FIFO-linearisations. Even then, though, there may be more firing sequences than what can be deduced from the description L of the observed behaviour. By Theorem 4.(2), a precise fit of the enterprise nets is preserved in the I-net constructed. We are currently working on a characterisation of properties of L that would guarantee that the language of L is exactly $\mathtt{lin}(L)$.

Example 8. (Ex. 7 ctd.) Let w and v be as in Example 7 and let $L = \{w\}$. Let $\mathcal{DA} = ([\Sigma_{IF}, \Sigma_{EX}], [\Sigma_{int}, \Sigma_{inp}, \Sigma_{out}], mt, cp)$ be the 2-DCA, with components $\mathcal{DA}_{IF}$ and $\mathcal{DA}_{EX}$, as given in Fig. 4, representing the actions available to the collaboration between the Investment firm and the Exchange from Example 7. Let $\mathcal{A}_E$ and $\mathcal{A}_\mathcal{I}$ be the E-net and I-net discovery algo-

	Σ_{int}	Σ_{inp}	Σ_{out}
Σ_{IF}	ii1,ii2	rc	so
Σ_{EX}	ei1,ei2	ro	sc

$mt(\mathbf{so}) = \boldsymbol{m}_o$
$mt(\mathbf{ro}) = \boldsymbol{m}_o$
$mt(\mathbf{sc}) = \boldsymbol{m}_c$
$mt(\mathbf{rc}) = \boldsymbol{m}_c$

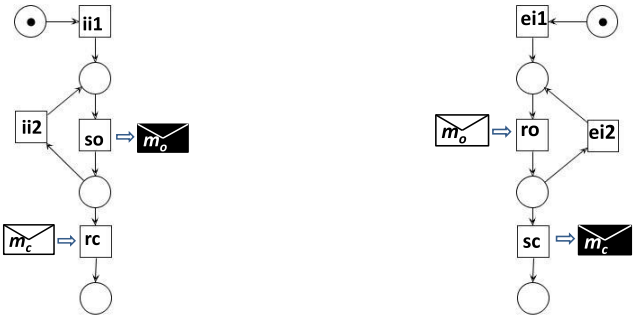
$cp(\mathbf{so}) = \boldsymbol{ro}$
$cp(\mathbf{ro}) = \boldsymbol{so}$
$cp(\mathbf{sc}) = \boldsymbol{rc}$
$cp(\mathbf{rc}) = \boldsymbol{sc}$

Fig. 4: The 2-DCA $\mathcal{DA}$

rithms respectively, derived from a process discovery algorithm $\mathcal{A}$. Furthermore, $\mathcal{A}_E(proj_{\Sigma_{IF}}(L), \mathcal{DA}_{IF}) = \mathcal{E}_{IF}$ and $\mathcal{A}_E(proj_{\Sigma_{EX}}(L), \mathcal{DA}_{EX}) = \mathcal{E}_{EX}$ are as depicted in Figs. 5.(a) and (b) respectively. Then $\mathcal{A}_\mathcal{I}(L, \mathcal{DA}) = \mathcal{I}(V, \varphi)$ is the I-net in Fig. 3. Since L satisfies the "if" conditions in Theorem 1, $L \subseteq \mathcal{L}(\mathcal{I}(V, \varphi))$. Moreover, as $v \in \mathtt{lin}_{\mathtt{FIFO}}(w)$ we have from Theorem 4, that also $v \in \mathcal{L}(\mathcal{I}(V, \varphi))$ □

7 Discussion

In this paper we have considered the problem of discovering a distributed process model (in the form of an I-net) from an event log (given in the form of a language). Moreover, the number of participating processes (modelled as E-nets) is given together with their channels (in the form of matching input and output actions). We have shown how an (existing) algorithm



(a) E-net $\mathcal{E}_{IF}$ (b) E-net $\mathcal{E}_{EX}$

Fig. 5

for the discovery of Petri net models from event logs of individual processes could be used in a new, distributed, algorithm to discover an I-net. Moreover, by Theorem 1, fitness of the I-net produced by the algorithm is guaranteed if and only if fitness of the component nets is guaranteed and the event log has the prefix property (a natural assumption for event logs of distributed processes composed of components that interact via asynchronous communication channels). It is interesting to reflect on the inclusion of these linearisations (see Corollary 2 and Theorem 4.(1)) and the role of the channels. The exchange of independent occurrences of actions suggests a relationship to the well-known Mazurkiewicz traces (equivalence classes of words, see, eg., [21]) and their dependence graphs (defining their causal structure in the form of a labelled partial order, see [15]). There is however an important difference: independence in Mazurkiewicz' theory is between actions rather than their occurrences. The independence between occurrences considered here can be seen as being context sensitive and determined by the history leading to these occurrences which leads to a theory of context dependent or local traces (see, eg., [10, 17]). In [14], a model similar to our I-nets is considered. There channel properties are investigated in terms of asynchronous I/O-transition systems rather than languages. The asynchronously composed I/O-Petri nets considered, use communication channels modelled by places of Petri nets rather than imposing the usual FIFO requirement of communication channels. However, as can be seen from our results, choosing FIFO channels or the standard unordered places, does not affect the language.

Returning to the discovering of distributed process models, it was our initial idea to investigate to what extent existing algorithms for the dis-

covery of single (isolated) processes could be used to discover distributed systems. The subject of process mining is an active research area, that has resulted in many process discovery algorithms (see, eg., [6] for an overview). Typically, these algorithms allow for the discovery of local processes in isolation, i.e., interaction between processes is not taken into account. Note that the approach as outlined here, is also different from the one followed in [2, 4] where the problem of process mining from large event logs is addressed. There, different options to distribute the mining problem over sublogs (with overlapping activities) are considered, whereas in the current paper we assume the (disjoint) components given. Of special interest are the passages from [3]. Provided a causal structure (a 'skeleton' of the process) is known, a larger model can be decomposed into (synchronising) fragments making a divide–and–conquer approach possible In [9] the event logs of Multi Agent Systems are projected onto each agent to discover component models in terms of workflow nets by using existing process discovery algorithms, similar to our approach. By means of α-morphisms an abstraction of each component model is derived. The aim of [9] however, is to show that if the composition of these abstract models is sound, the composition of the original component models is sound.

References

1. Wil M. P. van der Aalst, Arjan J. Mooij, Christian Stahl and Karsten Wolf (2009) Service Interaction: Patterns, Formalization, and Analysis Lect. Notes in Comput. Sci. 5569, 42–88.
2. Wil M. P. van der Aalst (2012) Distributed Process Discovery and Conformance Checking. Lect. Notes in Comput. Sci. 7212, 1–25.
3. Wil M. P. van der Aalst (2012) Decomposing Process Mining Problems Using Passages. Lect. Notes in Comput. Sci. 7347, 72–91.
4. Wil M. P. van der Aalst (2013) Decomposing Petri nets for Process Mining: A Generic Approach. Distr. and Parallel Databases 31(4), 471–507.
5. Wil M. P. van der Aalst (2013) Service Mining: Using Process Mining to Discover, Check, and Improve Service Behavior IEEE Trans. Serv. Comput. 6(4), 525–535.
6. Wil M. P. van der Aalst (2016) *Process Mining - Data Science in Action* Second Edition, Springer.
7. Wil M. P. van der Aalst and Christian Stahl (2011) *Modeling Business Processes - A Petri Net-Oriented Approach*, MIT Press.
8. Adriano Augusto, Raffaele Conforti, Marlon Dumas, Marcello La Rosa, Fabrizio Maria Maggi, Andrea Marrella, Massimo Mecella, and Allar Soo (2017) Automated Discovery of Process Models from Event Logs: Review and Benchmark. arXiv:1705.02288v3[cs.SE].

9. Luca Bernardinello, Irina A. Lomazova, Roman Nesterov and Lucia Pomello (2018) Compositional Discovery of Workflow Nets from Event Logs Using Morphisms. ATAED 2018, CEUR Workshop Proceedings 2115, 39-55.
10. Isabelle Biermann and Brigitte Rozoy (1997) Reliable Generalized and Context Dependent Commutation relations. Lect. Notes in Comput. Sci. 1214, 165–176.
11. Josep Carmona, Boudewijn F. van Dongen, Andreas Solti, and Matthias Weidlich (2018) *Conformance Checking - Relating Processes and Models*, Springer.
12. EDSN (2018) Marktfacilitering `https://www.edsn.nl/`
13. Luís Gomes and João Paulo Barros (2005) Structuring and composability issues in Petri nets modeling. IEEE Trans. Industrial Informatics 1(2), 112 – 123.
14. Serge Haddad, Rolf Hennicker, and Mikael H. Møller (2013) Channel Properties of Asynchronously Composed Petri Nets. Lect. Notes in Comput. Sci. 7927, 368 – 388.
15. Hendrik Jan Hoogeboom and Grzegorz Rozenberg (1995) Dependence graphs. In: Diekert, V., Rozenberg, G. (eds.) *The Book of Traces*, World Scientific, Singapore, 43–67.
16. HL7 (2015) Health Level Seven INTERNATIONAL `http://www.hl7.org/`
17. P.W. Hoogers, H.C.M.Kleijn, and P.S. Thiagarajan (1995) A trace semantics for Petri nets. Inf. Comput. 117(1), 98–114.
18. Pieter M. Kwantes and Jetty Kleijn (2018) On the Synthesis of Industry Level Process Models from Enterprise Level Process Models. ATAED 2018, CEUR Workshop Proceedings 2115, 6–22.
19. Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst (2013) Discovering Block-Structured Process Models from Event Logs - A Constructive Approach. Lect. Notes in Comput. Sci. 7927, 311–329.
20. S.W.I.F.T. (2015) ISO20022 Universal financial industry message scheme `http://www.iso20022.org`
21. Antoni W. Mazurkiewicz (1987) Trace theory. Lect. Notes in Comput. Sci. 255, 279–324.
22. Wolfgang Reisig (2018) Towards a conceptual foundation of service composition. Comput. Sci. Res. Dev. 33(3-4): 281-289
23. Wolfgang Reisig (2019) Associative composition of components with double-sided interfaces. Acta Inf. 56(3): 229-253
24. GS1US (2018) RosettaNet `http://www.rosettanet.org/`
25. Karsten Wolf (2009) Does My Service Have Partners? Trans. Petri Nets Other Model. Concurr. 2: 152-171 (2009)