Compositional Service Trees

Wil M.P. van der Aalst[1], Kees M. van Hee[1], Peter Massuthe[2], Natalia Sidorova[1], and Jan Martijn van der Werf[1]

[1] Technische Universiteit Eindhoven,
Department of Mathematics and Computer Science,
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{W.M.P.v.d.Aalst, k.m.v.hee,n.sidorova,j.m.e.m.v.d.werf}@tue.nl
[2] Humboldt-Universität zu Berlin, Institut für Informatik,
Unter den Linden 6, 10099 Berlin, Germany
massuthe@informatik.hu-berlin.de

Abstract. In the world of Service Oriented Architectures, one deals with networks of cooperating components. A component offers services; to deliver a service it possibly needs services of other components, thus forming a *service tree*. This tree is built dynamically and not known beforehand. It is hard to verify the behavior of a service tree by using standard verification techniques, because these techniques typically assume a static flattened model. In this paper we model a component by an open Petri net. We give a sufficient condition for proper completion (called soundness) that requires only pairwise checks of the service compositions. We also provide a correctness-by-construction approach for building services trees.

1 Introduction

According to the paradigm of Service Oriented Architectures (SOA) a system can be seen as a (possibly open) network of cooperating *components* based on asynchronous communication [3, 8, 18]. A component offers *services* to a *client* which may be a component itself. Each component may play two roles: service provider and service client. One of the interesting features of SOA is the *dynamic binding* of services: in order to provide a service S for its client, a component may invoke a service S' of another component during the execution, while it might be not known at the beginning of this execution that the service S' was needed and which component would be selected to deliver it. In this way, the components form a tree, called a *service tree*, to *deliver* a certain service. However, this tree is not known to any party and for privacy and security reasons we often do not want the tree to be known to anybody. This makes the verification of behavioral correctness very hard.

Correctness requirements concern both dataflow and control flow. In this paper, we focus on the control flow aspect, i.e., on the *orchestration*. There are several approaches to define the orchestration process. BPEL (Business process Execution Language) is one of the main languages to specify the orchestration

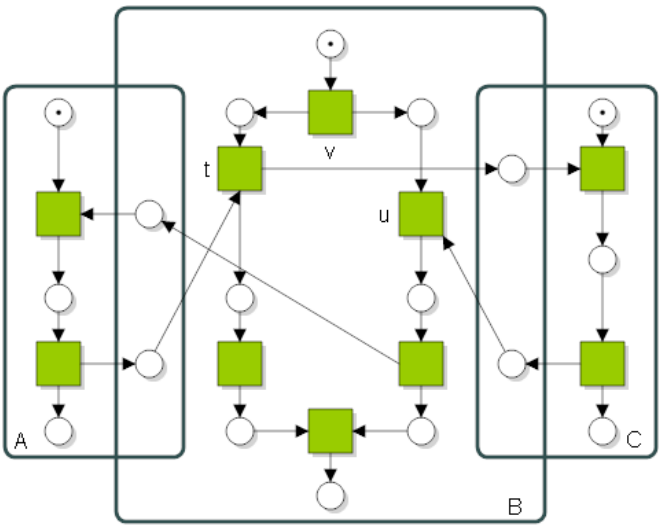

Fig. 1. The composition of three composable OWNs

at the implementation level [9]. We use here Petri nets for modeling the orchestration of components in order to analyse the behavioral properties.

Due to dynamic binding, we never know how a service tree will look like, and therefore, we are not able to check a complete service tree beforehand. Thus, if we want ensure proper termination we need a verification method that only considers pairwise compositions of components. Therefore, we propose a *correctness-by-construction* method that guarantees that if each pair of connected components satisfies some *correctness condition*, then the whole service tree will be sound.

We model services and service trees as *open Petri nets* (OPNs), i.e., a Petri net with a set of interface (input or output) places [5, 15, 18]. An OPN has one initial marking and one final marking, which is usually a deadlock. A composition of two OPNs is an OPN again; the corresponding interface places of the two nets are fused and they are not a part of the interface of the resulting net anymore. Sometimes, we consider a more restrictive class of OPNs, called *open workflow nets* (OWNs) which have one initial place, one final place, and an arbitrary number of interface places. Both OPNs and OWNs can be seen as a generalization of the classical workflow nets [1, 10].

The behavioral correctness criterion we consider is the *weak termination* property of services, which can be seen as generalization of the *soundness* concept of workflow nets [1], and therefore we also call it soundness. A stand-alone OPN is called *sound* if for each marking reachable from the initial marking the final marking is reachable, discarding the interface places.

We illustrate the need for compositional soundness conditions for service trees on the example shown in Figure 1. Here, we have three components, A, B, and C. The composition of A with B is sound, as well as the composition of B with C. However, the composition of the three has a deadlock, since this composition introduces a cyclic dependency implying a deadlock.

We will consider two approaches: a *posteriori* approach and a *constructive* approach. In the posteriori approach for each pair of composed components in a service tree we have to verify the correctness condition by checking a specific *simulation* relation. In the constructive approach we apply stepwise refinement. We start with a service tree of two composed components, which are known to satisfy the correctness condition. Then we select two so-called *synchronized* places and we refine them simultaneously by correctly composed components. We can also extend the tree by adding new leaves to it at arbitrary nodes.

In Section 2 we give basic definitions. In Section 3 we introduce the basic concepts of OPNs. In Section 4 we define the composition operator for OPNs and give sufficient conditions for soundness of service trees. In Section 5 we introduce the stepwise refinement approach in general and in Section 6 we present a correct-by-construction method for service trees. Finally, we conclude with related and future work in Section 7.

2 Preliminaries

Let S be a (finite) set. The powerset of S is denoted by $\mathcal{P}(S) = \{S' \mid S' \subseteq S\}$. We use $|S|$ for the number of elements in S. Two sets U and V are *disjoint* if $U \cap V = \emptyset$. A *bag* m over S is a function $m : S \to \mathbb{N}$. We denote e.g. the bag m with an element a occurring once, b occurring three times and c occurring twice by $m = [a, b^3, c^2]$. The set of all bags over S is denoted by $\mathbb{N}^S$. Sets can be seen as a special kind of bag were all elements occur only once. We use $+$ and $-$ for the sum and difference of two bags, and $=, <, >, \leq, \geq$ for the comparison of two bags, which are defined in a standard way. The projection of a bag $m \in \mathbb{N}^S$ on elements of a set $U \subseteq S$, is denoted by $m_{|U}$, and is defined by $m_{|U}(u) = m(u)$ for all $u \in U$ and $m_{|U}(u) = 0$ for all $u \in S \setminus U$. Furthermore, if for some $n \in \mathbb{N}$, disjoint sets $U_i \subseteq S$ with $1 \leq i \leq n$ exist such that $S = \bigcup_{i=1}^{n} U_i$, then $m = \sum_{i=1}^{n} m_{|U_i}$.

A *sequence* over S of length $n \in \mathbb{N}$ is a function $\sigma : \{1, \ldots, n\} \to S$. If $n > 0$ and $\sigma(i) = a_i$ for $i \in \{1, \ldots, n\}$, we write $\sigma = \langle a_1, \ldots, a_n \rangle$, and σ_i for $\sigma(i)$. The length of a sequence is denoted by $|\sigma|$. The sequence of length 0 is called the *empty sequence*, and is denoted by ϵ. The set of all finite sequences over S is denoted by S^*. Let $\nu, \gamma \in S^*$ be two sequences. *Concatenation*, denoted by $\sigma = \nu; \gamma$ is defined as $\sigma : \{1, \ldots, |\nu| + |\gamma|\} \to S$, such that for $1 \leq i \leq |\nu|$: $\sigma(i) = \nu(i)$, and for $|\nu| + 1 \leq i \leq |\nu| + |\gamma|$: $\sigma(i) = \gamma(i - |\nu|)$. A projection of a sequence $\sigma \in S^*$ on elements of a set $U \subseteq S$ (i.e. eliminating the elements from $S \setminus U$) is denoted as $\sigma_{|U}$.

If we give a tuple a name, we subscript the elements with the name of the tuple, e.g. for $N = (A, B, C)$ we refer to its elements by A_N, B_N, and C_N. If the context is clear, we omit the subscript.

Definition 1 (Labeled transition system). *A* labeled transition system *(LTS) is a* 5*-tuple* $(S, \mathcal{A}, \longrightarrow, s_i, s_f)$ *where*

- S *is a set of* states*;*
- $\mathcal{A}$ *is a set of* actions*;*
- $\longrightarrow \subseteq S \times (\mathcal{A} \cup \{\tau\}) \times S$ *is a* transition relation*, where* $\tau \notin \mathcal{A}$ *is the silent action [10].*
- $s_i \in S$ *is the initial state;*
- $s_f \in S$ *is the final state.*

For $m, m' \in S$ and $a \in \mathcal{A}$, we write $L : m \xrightarrow{a} m'$ if and only if $(m, a, m') \in \longrightarrow$. If the context is clear, we omit the L. A state $m \in S$ is called a *deadlock* if no action $a \in \mathcal{A} \cup \{\tau\}$ and state $m' \in S$ exist such that $m \xrightarrow{a} m'$. We often require that s_f is a deadlock.

We define $\Longrightarrow$ as the smallest relation such that $m \Longrightarrow m'$ if $m = m' \vee \exists m'' \in S : m \Longrightarrow m'' \xrightarrow{\tau} m'$. $m \Longrightarrow m'$ is sometimes also referred to as $m \xRightarrow{\tau} m'$, i.e., a path of zero or more silent actions [10]. We define $\xRightarrow{a}$ as the smallest relation such that $m \xRightarrow{a} m'$ if $\exists m'' \in S : m \Longrightarrow m'' \xrightarrow{a} m'$.

Definition 2 (Hiding). *Let* $L = (S, \mathcal{A}, \longrightarrow, s_i, s_f)$ *be an LTS. Let* $H \subseteq \mathcal{A}$*. We define the operation* τ_H *on an LTS by* $\tau_H(L) = (S, \mathcal{A} \setminus H, \longrightarrow', s_i, s_f)$*, where for* $m, m' \in S$ *and* $a \in \mathcal{A}$ *we have* $(m, a, m') \in \longrightarrow'$ *if and only if* $(m, a, m') \in \longrightarrow$ *and* $a \notin H$ *and* $(m, \tau, m') \in \longrightarrow'$ *if and only if* $(m, \tau, m') \in \longrightarrow$ *or* $(m, a, m') \in \longrightarrow$ *and* $a \in H$*.*

We define simulation and bisimulation relations on labeled transition systems.

Definition 3 (Simulation, bisimulation). *Let* $L = (S, \mathcal{A}, \longrightarrow, s_i, s_f)$ *and* $L' = (S', \mathcal{A}, \longrightarrow', s_i', s_f')$ *be two LTSs. The relation* $R \subseteq S \times S'$ *is a* simulation*, denoted by* $L \preceq_R L'$*, if:*

1. $s_i \, R \, s_i'$ *and* $s_f \, R \, s_f'$*;*
2. $\forall m, m' \in S, \bar{m} \in S', a \in \mathcal{A} \cup \{\tau\} : (m \xrightarrow{a} m' \wedge m \, R \, \bar{m}) \Rightarrow (\exists \bar{m}' \in S' : \bar{m} \xRightarrow{a}{}' \, \bar{m}' \wedge m' \, R \, \bar{m}')$.
3. $\forall m' \in S' : (s_f \, R \, m') \Rightarrow (m' \Longrightarrow s_f')$.

If both R *and* R^{-1} *are simulations,* R *is a* bisimulation*.*

Note that $L \preceq_R L'$ means that L' can mimic L, i.e., L' is able to *simulate* L.

Petri nets A *Petri net* is a 3-tuple $N = (P, T, F)$ where (1) P and T are two disjoint sets of *places* and *transitions* respectively; (2) $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*. The elements from the set $P \cup T$ are called the *nodes* of N. Elements of F are called *arcs*. Places are depicted as circles, transitions as squares. For each element $(n_1, n_2) \in F$, an arc is drawn from n_1 to n_2. Two Petri nets $N = (P, T, F)$ and $N' = (P', T', F')$ are *disjoint* if and only if $(P \cup T) \cap (P' \cup T') = \emptyset$. Let $N = (P, T, F)$ be a Petri net. Given a node $n \in (P \cup T)$, we define its *preset* ${}^{N}\bullet n = \{n' \mid (n', n) \in F\}$, and its *postset* $n{}^{N}\bullet = \{n' \mid (n, n') \in F\}$. We lift the notation of preset and postset to sets. Given a set $U \subseteq (P \cup T)$, ${}^{N}\bullet U = \bigcup_{n \in U} {}^{N}\bullet n$ and $U{}^{N}\bullet = \bigcup_{n \in U} n{}^{N}\bullet$. If the context is clear, we omit the N in the superscript.

Markings are states of a net. A *marking* m of a Petri net $N = (P, T, F)$ is defined as a bag over P. A pair (N, m) is called a *marked Petri net*. A transition $t \in T$ is enabled in a marking $m \in \mathbb{N}^P$ if and only if ${}^\bullet t \leq m$. An enabled transition may *fire*. A transition firing results in a new marking m' with $m' = m - {}^\bullet t + t^\bullet$, denoted by $N : m \xrightarrow{t} m'$. If the context is clear, we omit the N. A sequence $\sigma = \langle t_1, \ldots, t_n \rangle$ is a *firing sequence* of (N, m) if and only if there exist markings $m_1, \ldots, m_n \in \mathbb{N}^P$ such that $N : m \xrightarrow{t_1} m_1 \xrightarrow{t_2} \ldots m_{n-1} \xrightarrow{t_n} m_n$. We write $N : m \xrightarrow{*} m'$ if there exists a (possibly empty) firing sequence $\sigma \in T^*$ such that $N : m \xrightarrow{\sigma} m'$. The set of all reachable markings of a marked Petri net (N, m) is denoted by $\mathcal{R}(N, m) = \{m' \in \mathbb{N}^P \mid N : m \xrightarrow{*} m'\}$.

A place $p \in P$ is *safe* if for any marking $m' \in \mathcal{R}(N, m)$, holds that $m'(p) \leq 1$. A marked Petri net is called safe if all its places are safe. The marking m is a *deadlock* if there exists no transition $t \in T$ such that ${}^\bullet t \leq m$.

Definition 4 (Workflow net, soundness). *A Petri net $N = (P, T, F)$ is called a* workflow net *if (1) there exists exactly one place $i \in P$, called the* initial place*, such that ${}^\bullet i = \emptyset$, (2) there exists exactly one place, called the* final place*, $f \in P$ such that $f^\bullet = \emptyset$, and (3) all nodes are on a path from i to f. N is* sound *if $[f] \in \mathcal{R}(N, m)$ for any marking $m \in \mathcal{R}(N, [i])$.*

3 Open Petri Nets

The SOA paradigm builds upon asynchronous communications: a *component* sends messages to communicate with other components. In this approach, we model a component by a Petri net. Communication is done through an interface, modeled by input and output places. We call such a Petri net with an interface an *open Petri net* (OPN) [5, 15, 18].

Definition 5 (Open Petri net). *An* open Petri net *(OPN) is a 7-tuple (P, I, O, T, F, i, f) where*

- *$(P \cup I \cup O, T, F)$ is a Petri net;*
- *P is a set of* internal places*;*
- *I is a set of* input places*, and ${}^\bullet I = \emptyset$;*

– O *is a set of* output places*, and* $O^\bullet = \emptyset$*;*
– P*,* I*,* O*, and* T *are pairwise disjoint;*
– $i \in \mathbb{N}^P$ *is the* initial marking*,*
– $f \in \mathbb{N}^P$ *is the* final marking*, and*
– f *is a deadlock.*

We call the set $I \cup O$ *the* interface places *of the OPN. Two OPNs* N *and* M *are called* disjoint *if* P_N*,* P_M*,* I_N*,* I_M*,* O_N*,* O_M*,* T_N *and* T_M *are pairwise disjoint.*

Note that the initial and final markings cannot mark interface places. Although we allow interface places to have more than one connected transition, it is always possible to transform the nets to equivalent ones with exactly one connected transition for interface places (cf. [4]).

In order to inspect and refer to the internal working of a component, ignoring the communication aspects, we introduce the notion of a skeleton. The skeleton is the Petri net of the component without any interface places.

Definition 6 (Skeleton). *Let* N *be an OPN. The* skeleton *of* N *is defined as the Petri net* $\mathcal{S}(N) = (P_N, T_N, F)$ *with* $F = F_N \cap ((P_N \times T_N) \cup (T_N \times P_N))$*. We use* $\mathcal{R}(N)$ *as a shorthand notation for* $\mathcal{R}(\mathcal{S}(N), i_N)$*.*

The semantics of an OPN is given by its LTS.

Definition 7 (LTS of an OPN). *Let* N *be an OPN. Its labeled transition system is defined as:* $\mathcal{T}(N) = (\mathcal{R}(N), T, \longrightarrow, i_N, f_N)$ *with* $(m, t, m') \in \longrightarrow$ *if and only if* $\mathcal{S}(N) : m \xrightarrow{t} m'$*.*

We focus on services that try to reach a goal. We therefore introduce the notion of an open workflow net, i.e., an open Petri net such that the skeleton is a workflow, and only the initial place is marked in the initial marking.

Definition 8 (Open workflow net). *Let* N *be an OPN. It is called an* open workflow net *(OWN) iff its skeleton is a workflow net with initial place* $i \in P$ *and final place* $f \in P$*, such that* ${}^\bullet i = \emptyset$*,* $f^\bullet = \emptyset$*,* $i_N = [i]$*,* $f_N = [f]$*.*

Note that in an OWN N, the final marking f_N is always a deadlock. All transitions and internal places lie on a path from i to f, since the skeleton is a workflow, whereas interface places cannot not have this property. We would like services to be sound, i.e., always have the possibility to terminate properly. As the open Petri net has interface places, termination depends on the communication partners of the net. Still, we want to express that at least the service disregarding the communication is modelled in a proper way. Therefore, we define soundness on the skeleton of the open Petri net.

Definition 9 (Soundness of OPNs). *An OPN* N *is called* sound *if for any marking* $m \in \mathcal{R}(N)$ *we have* $\mathcal{S}(N) : m \xrightarrow{*} f_N$*.*

Note that if an OPN N is sound and f_N is a nonempty deadlock, then the initial marking i_N cannot be the empty marking, since if f is reachable from the empty marking, also $2 \cdot f$ is reachable, but f cannot be reached from $2 \cdot f$ thus invalidating soundness.

4 Composition

Two open Petri nets can be composed by fusing interface places with the same name.

Definition 10 (Composition). *Let A and B be two OPNs. Their* composition *is an OPN $A \oplus B = (P, I, O, T, F, i, f)$ defined by:*

- $P = P_A \cup P_B \cup (I_A \cap O_B) \cup (I_B \cap O_A)$;
- $I = (I_A \setminus O_B) \cup (I_B \setminus O_A)$;
- $O = (O_A \setminus I_B) \cup (O_B \setminus I_A)$;
- $T = T_A \cup T_B$;
- $F = F_A \cup F_B$;
- $i = i_A + i_B$;
- $f = f_A + f_B$.

Two OPNs are composable if they do not share any internal places, input places and output places. Note that an input place of one net can be an output place of another net.

Definition 11 (Composable). *Two OPNs A and B are* composable *if and only if $(P_A \cup I_A \cup O_A \cup T_A) \cap (P_B \cup I_B \cup O_B \cup T_B) = (I_A \cap O_B) \cup (O_A \cap I_B)$.*

The composition operator is commutative and associative for composable OPNs.

Lemma 12 (Commutativity and associativity of composition). *Let A, B and C be three pairwise composable OPNs. Then $A \oplus B = B \oplus A$ and $(A \oplus B) \oplus C = A \oplus (B \oplus C)$. In addition, if $\mathcal{O} = (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$, we have $A \oplus \mathcal{O} = \mathcal{O} \oplus A = A$.*

Proof. Follows directly from the definition of $\oplus$. □

Using the composition notion, we define a projection relation and simulation based on this relation.

Definition 13 (Projection relation, simulation property). *Let A and B be two composable OPNs. The* projection relation $R \subseteq \mathbb{N}^{P_A} \times \mathbb{N}^{P_{A \oplus B}}$ *is defined as $R = \{(m_{|P_A}, m) \mid m \in \mathcal{R}(A \oplus B)\}$. We use the shorthand notation $A \oplus B \succeq A$ for $\tau_{T_B}(\mathcal{T}(A \oplus B)) \succeq_R \mathcal{T}(A)$. If $A \oplus B \succeq A$, we say that the composition has the* simulation property*.*

Note that the notation $A \oplus B \succeq A$ states that the projection relation is a simulation. If $A \oplus B \succeq A$, then $A \oplus B$ is able to mimic the behavior of A after abstracting away the transitions in B.

The next lemma shows that for a sequence σ in $A \oplus B$ and a sequence $\tilde{\sigma}$ in $B \oplus C$, such that their projections on B are identical, a composed sequence $\overline{\sigma}$ exists, such that the projection of $\overline{\sigma}$ on A and B is identical to σ, and the projection of $\overline{\sigma}$ on B and C is identical to $\tilde{\sigma}$.

Lemma 14 (Combining firing sequences). *Let A, B and C be three pairwise composable OPNs and let A and C have disjoint interface places. Let $m \in \mathcal{R}(A \oplus B \oplus C)$, $\sigma \in (T_A \cup T_B)^*$, $\tilde{\sigma} \in (T_B \cup T_C)^*$, $m' \in \mathcal{R}(A \oplus B)$ and $m'' \in \mathcal{R}(B \oplus C)$ such that $A \oplus B : m_{|P_{A\oplus B}} \xrightarrow{\sigma} m'$, $B \oplus C : m_{|P_{B\oplus C}} \xrightarrow{\tilde{\sigma}} m''$ and $\sigma_{|T_B} = \tilde{\sigma}_{|T_B}$. Then there exist a firing sequence $\overline{\sigma} \in (T_A \cup T_B \cup T_C)^*$ such that $\overline{\sigma}_{|T_A \cup T_B} = \sigma$ and $\overline{\sigma}_{|T_B \cup T_C} = \tilde{\sigma}$, and a marking $\overline{m} \in \mathcal{R}(A \oplus B \oplus C)$ such that $A \oplus B \oplus C : m \xrightarrow{\overline{\sigma}} \overline{m}$, $\overline{m}_{|P_{A\oplus B}} = m'$ and $\overline{m}_{|P_{B\oplus C}} = m''$.*

Proof. We prove this lemma by induction on the length of $\sigma_{|T_B}$. If $\sigma_{|T_B} = \epsilon$, then σ can execute independently of $\tilde{\sigma}$, since transitions in A and C have no places in common. Hence, the statement holds for length 0.

Let it hold for length n and we consider $m \in \mathcal{R}(A \oplus B \oplus C)$, $\sigma \in (T_A \cup T_B)^*$, $\tilde{\sigma} \in (T_B \cup T_C)^*$, $m' \in \mathcal{R}(A \oplus B)$ and $m'' \in \mathcal{R}(B \oplus C)$ such that $A \oplus B : m_{|P_{A\oplus B}} \xrightarrow{\sigma} m'$, $B \oplus C : m_{|P_{B\oplus C}} \xrightarrow{\tilde{\sigma}} m''$, with $\sigma_{|T_B} = \tilde{\sigma}_{|T_B} = \sigma'; b$ for some $b \in T_B$ and $|\sigma'| = n$. Then $\sigma = \sigma_1; b; \sigma_2$ for some $\sigma_1 \in (T_A \cup T_B)^*$, $\sigma_2 \in (T_A)^*$ and $\tilde{\sigma} = \tilde{\sigma}_1; b; \tilde{\sigma}_2$ for some $\tilde{\sigma}_1 \in (T_B \cup T_C)^*$ and $\tilde{\sigma}_2 \in (T_C)^*$ and $\sigma_{1|T_B} = \tilde{\sigma}_{1|T_B} = \sigma'_{|T_B}$. By the induction hypothesis there exists $\overline{\sigma}_1$ such that $\overline{\sigma}_{1|T_A \cup T_B} = \sigma_1$ and $\overline{\sigma}_{1|T_B \cup T_C} = \tilde{\sigma}_1$ and a marking $\overline{m}_1$ such that $A \oplus B \oplus C : m \xrightarrow{\overline{\sigma}_1} \overline{m}_1$, $\overline{m}_{1|P_{A\oplus B}} = m'_1$ and $\overline{m}_{1|P_{B\oplus C}} = m''_1$.

We have $A \oplus B \oplus C : \overline{m}_1 \xrightarrow{b} \overline{m}_2$, since b is enabled both in $A \oplus B$ and in $B \oplus C$. In the resulting marking $\overline{m}_2$, σ_2 and $\tilde{\sigma}_2$ are independently enabled, since A and C have no common places. Hence, we can apply the induction hypothesis on σ_2 and $\tilde{\sigma}_2$ which gives a firing sequence $\overline{\sigma}_2$ such that $\overline{\sigma}_{2|T_A \cup T_B} = \sigma_2$ and $\overline{\sigma}_{2|T_B \cup T_C} = \tilde{\sigma}_2$ and a marking $\overline{m}$ such that $A \oplus B \oplus C : \overline{m}_2 \xrightarrow{\overline{\sigma}_2} \overline{m}$, $\overline{m}_{|P_{A\oplus B}} = m'$ and $\overline{m}_{|P_{B\oplus C}} = m''$. Hence, $\overline{\sigma} = \overline{\sigma}_1; b; \overline{\sigma}_2$ has the desired property.

□

Lemma 15. *Let A and B be two composable OPNs. Let $m, m' \in \mathcal{R}(A \oplus B)$ and $\sigma \in (T_A \cup T_B)^*$ such that $A \oplus B : m \xrightarrow{\sigma} m'$. Then $A : m_{|P_A} \xrightarrow{\sigma_{|T_A}} m'_{|P_A}$ and $B : m_{|P_B} \xrightarrow{\sigma_{|T_B}} m'_{|P_B}$.*

Since a service tree is not known in advance, we need a condition such that if any two composed services satisfy this condition, the whole service tree is sound. The example in Figure 1 shows that soundness itself is not the right property, since although both $A \oplus B$ and $B \oplus C$ are sound, the composition of the three, $A \oplus B \oplus C$ is not. The reason is that $A \oplus B$ and $B \oplus C$ are acyclic but $A \oplus B \oplus C$ has a cycle which causes a deadlock. This example shows how tricky the asynchronous composition of OPNs is. We use Lemma 14 to come to a sufficient condition for soundness of the composition of three OPNs, by only pairwise checking of the connections between the OPNs.

If a component B is composed with A, and this composition is sound, composing it with C should not destroy soundness. This can occur e.g. if C blocks some execution path of B. We will show that a sufficient condition for the soundness of $A \oplus B \oplus C$ is: C should not block any execution of B that starts in the initial marking and ends in the final marking.

First, we formally define the sufficient condition.

Definition 16 (Condition $\Omega_{A,B}$). *Let A and B be two composable OPNs. Condition $\Omega_{A,B}$ holds if and only if* $\forall m \in \mathcal{R}(A \oplus B),\ \sigma \in (T_A)^* : (A : m_{|P_A} \xrightarrow{\sigma} f_A) \ \Rightarrow\ (\exists \tilde{\sigma} \in (T_A \cup T_B)^* : (A \oplus B : m \xrightarrow{\tilde{\sigma}} f_A + f_B) \ \wedge\ \tilde{\sigma}_{|T_A} = \sigma)$.

Note that we do not require soundness of A or B here, but if A is sound, $\Omega_{A,B}$ implies that $A \oplus B$ is sound.

Lemma 17. *Let A and B be two composable OPNs. If A is sound and condition $\Omega_{A,B}$ holds, then $A \oplus B$ is sound.*

Now we show that condition $\Omega_{B,C}$ is a sufficient condition allowing to extend an arbitrary composition $A \oplus B$ by C obtaining a sound $A \oplus B \oplus C$.

Theorem 18 (Sufficient condition). *Let A, B and C be three pairwise composable OPNs such that A and C have disjoint interface places. If the composition $A \oplus B$ is sound and $\Omega_{B,C}$ holds, then $A \oplus B \oplus C$ is sound.*

Proof. Let $m \in \mathcal{R}(A \oplus B \oplus C)$. Then, by Lemma 15, $m_{|P_{A\oplus B}} \in \mathcal{R}(A \oplus B)$ and $m_{|P_{B\oplus C}} \in \mathcal{R}(B \oplus C)$. By the soundness of $A \oplus B$ there is a $\sigma \in (T_{A\oplus B})^*$ such that $A \oplus B : m_{|P_{A\oplus B}} \xrightarrow{\sigma} f_A + f_B$. By condition $\Omega_{B,C}$ applied to $m_{|P_{B\oplus C}}$ and $\sigma_{|T_B}$, there is a $\tilde{\sigma} \in (T_B \cup T_C)^*$ such that $\tilde{\sigma}_{|T_B} = \sigma_{|T_B}$ and $B \oplus C : m_{|P_{B\oplus C}} \xrightarrow{\tilde{\sigma}} f_B + f_C$. By Lemma 14, there exist a firing sequence $\overline{\sigma} \in (T_A \cup T_B \cup T_C)^*$ and marking $\overline{m}$ such that $\overline{\sigma}_{|T_A \cup T_B} = \sigma$ and $\overline{\sigma}_{|T_B \cup T_C} = \tilde{\sigma}$ and $A \oplus B \oplus C : m \xrightarrow{\overline{\sigma}} \overline{m}$, $\overline{m}_{|P_{A\oplus B}} = f_A + f_B$ and $\overline{m}_{|P_{B\oplus C}} = f_B + f_C$. Note that the interface places between A and B and between B and C are empty. Since there are no interface places between A and C, we have $\overline{m} = f_A + f_B + f_C$. □

To facilitate the check of condition $\Omega_{A,B}$, we prove that $\Omega_{A,B}$ is equivalent to the condition "$A \oplus B$ simulates A".

Theorem 19 (Equivalent condition). *Let A and B be two composable OPNs and A be sound. Then $\Omega_{A,B}$ holds if and only if $A \oplus B \succeq A$.*

Proof. ($\Rightarrow$) Assume that $\Omega_{A,B}$ holds. We show that the projection relation $R = \{(m_{|P_A}, m) \mid m \in \mathcal{R}(A \oplus B)\}$ is a simulation.

1) By definition of $\oplus$, we have $i_A \, R \, i_{A\oplus B}$ and $f_A \, R \, f_{A\oplus B}$ provided that $f_{A\oplus B} \in \mathcal{R}(A \oplus B)$. The latter condition follows from $\Omega_{A,B}$.

2) Let $\overline{m} \in \mathcal{R}(A \oplus B)$ and $m = \overline{m}_{|P_A}$, i.e., $m \, R \, \overline{m}$. Note that by Lemma 15, $m \in \mathcal{R}(A)$. Further let $A : m \xrightarrow{t} m'$ for some $m' \in \mathcal{R}(A)$, $t \in T_A$.

Since A is sound, there exists a sequence $\sigma \in (T_A)^*$ such that $A : m \xrightarrow{t;\sigma} f_A$. By the condition $\Omega_{A,B}$, there is a sequence $\tilde{\sigma} \in (T_A \cup T_B)^*$, such that $A \oplus B : \overline{m} \xrightarrow{\tilde{\sigma}} f_A + f_B$ and $\tilde{\sigma}_{|T_A} = t;\sigma$. Thus, $A \oplus B : \overline{m} \xrightarrow{\tilde{\sigma}'} \overline{m}'' \xrightarrow{t} \overline{m}'$ (i.e. $A \oplus B : \overline{m} \xRightarrow{t} \overline{m}'$) for some markings $\overline{m}', \overline{m}'' \in \mathcal{R}(A \oplus B)$ and sequence $\tilde{\sigma}' \in (T_B)^*$. Since $\tilde{\sigma}' \in (T_B)^*$, for all places $p \in P_A$ holds $m(p) = \overline{m}(p) = \overline{m}''(p)$ and

$m'(p) = m(p) - {}^\bullet t(p) + t^\bullet(p) = \overline{m}''(p) - {}^\bullet t(p) + t^\bullet(p) = \overline{m}'(p)$. Thus $\overline{m}'_{|P_A} = m'$, and therefore $m' \, R \, \overline{m}'$.

3) The only reachable deadlock possible in A is f_A. Let $m \in \mathcal{R}(A \oplus B)$ such that $f_A \, R \, m$. No transition $t \in T_A$ is enabled in marking f_A, and hence, also not in m. By applying $\Omega_{A,B}$ to ϵ and m, we conclude that there exists a sequence $\tilde{\sigma} \in (T_A \cup T_B)^*$ such that $A \oplus B : m \xrightarrow{\tilde{\sigma}} f_A + f_B$. Since no transition of A can fire, $\tilde{\sigma} \in (T_B)^*$. Hence $A \oplus B : m \Longrightarrow f_A + f_B$.

($\Leftarrow$) Assume $A \oplus B \succeq A$. Let $m \in \mathcal{R}(A \oplus B), \sigma \in (T_A)^*$ such that $A : m_{|P_A} \xrightarrow{\sigma} f_A$. We prove by induction on the length of σ that $\exists \tilde{\sigma} \in (T_A \cup T_B)^* : (A \oplus B : m \xrightarrow{\tilde{\sigma}} f_A + f_B) \ \wedge \ \tilde{\sigma}_{|T_A} = \sigma$ to show that $\Omega_{A,B}$ holds.

Suppose $\sigma = \epsilon$, i.e. $m_{|P_A} \xrightarrow{\epsilon} f_A$, which implies $f_A = m_{|P_A}$ and thus $f_A \, R \, m$. Since $A \oplus B \succeq A$, there exists a sequence $\tilde{\sigma} \in (T_B)^*$ such that $A \oplus B : m \xrightarrow{\tilde{\sigma}} f_A + f_B$ with $\tilde{\sigma}_{|T_A} = \epsilon$.

Suppose $\sigma = t; \sigma'$, $t \in T_A$. Then, a marking $m' \in \mathcal{R}(A)$ exists, such that $A : m_{|P_A} \xrightarrow{t} m' \xrightarrow{\sigma'} f_A$. Since $A \oplus B \succeq A$, there exists a marking $\overline{m}' \in \mathcal{R}(A \oplus B)$ such that $A \oplus B : m \stackrel{t}{\Longrightarrow} \overline{m}'$ and $m' \, R \, \bar{m}'$. Hence, there is a marking $\overline{m}'' \in \mathcal{R}(A \oplus B)$ and a sequence $\tilde{\sigma}' \in (T_B)^*$ such that $A \oplus B : m \xrightarrow{\tilde{\sigma}'} \overline{m}'' \xrightarrow{t} \overline{m}'$. The induction hypothesis applied to σ' implies the existence of a sequence $\tilde{\sigma}'' \in (T_A \cup T_B)^*$ such that $\tilde{\sigma}''_{|T_A} = \sigma'$ and $A \oplus B : \overline{m}' \xrightarrow{\tilde{\sigma}''} f_A + f_B$. Hence, $\tilde{\sigma} = \tilde{\sigma}'; t; \tilde{\sigma}''$ is a sequence such that $A \oplus B : m \xrightarrow{\tilde{\sigma}} f_A + f_B$ and $\tilde{\sigma}_{|T_A} = \sigma$. □

We can extend our results to compositions that are more complex than chains of three components. A *service tree* is a tree of components connected to each other such that the higher OPNs can only "subcontract" work to lower level OPNs. The structure of the tree is defined by the tree function c. Each node i is an OPN representing a component that is delivering a service to its parent $c(i)$ using services of its children $c^{-1}(i)$. In the remainder of this section, we show that the sufficient condition is enough to only pairwise check the connections in the tree to decide whether the whole service tree is sound.

Definition 20 (Service tree). *Let $A_1, \ldots, A_n$ be pairwise composable OPNs. Let $c : \{2, \ldots, n\} \to \{1, \ldots, n-1\}$ be such that:*

- $\forall i \in \{2, \ldots, n\} : c(i) < i$,
- $\forall 1 \leq i < j \leq n : i \neq c(j) \ \Rightarrow \ I_{A_i} \cap O_{A_j} = \emptyset \ \wedge \ O_{A_i} \cap I_{A_j} = \emptyset$, *and*
- $\forall 1 \leq i < j \leq n : i = c(j) \ \Rightarrow \ I_{A_i} \cap O_{A_j} \neq \emptyset \ \vee \ O_{A_i} \cap I_{A_j} \neq \emptyset$.

We call $A_1 \oplus \ldots \oplus A_n$ a service tree *with root A_1 and tree function c.*

Lemma 17 together with Theorem 19 implies that if $B \oplus C \succeq B$ and B is sound, the composition is sound as well. Hence, if we combine the results so far, we can show that if the root of a service tree is sound, and all the connections fulfill the Ω condition, the whole service tree is sound.

Theorem 21 (Soundness of service trees). *Let $A_1, \ldots A_n$ be a service tree with root A_1 and tree function c. Further, let A_1 be sound and for $2 \leq i \leq n$, it holds that $A_i \oplus A_{c(i)} \succeq A_{c(i)}$. Then $A_1 \oplus \ldots \oplus A_n$ is sound.*

Proof. We prove this by induction on n. If $n = 1$, it is true by definition. Suppose it is true for $n = k - 1$. Let $n = k$. By the induction hypothesis: $A_1 \oplus \ldots \oplus A_{k-1}$ is sound and always $c(k) < k$. By the associativity and commutativity of $\oplus$ we have that $(A_1 \oplus \ldots \oplus A_{c(k)-1} \oplus A_{c(k)+1} \oplus \ldots \oplus A_{k-1}) \oplus A_{c(k)}$ is sound. We also have $A_k \oplus A_{c(k)} \succeq A_{c(k)}$. By Lemma 17 we have $(A_1 \oplus \ldots \oplus A_{c(k)-1} \oplus A_{c(k)+1} \oplus \ldots \oplus A_{k-1}) \oplus A_{c(k)} \oplus A_k$ is sound. Again by associativity and commutativity, the theorem is proven. □

5 Stepwise Refinement

In the previous section we showed that we can build sound service trees compositionally. In this section we present a construction method that guarantees condition Ω with simultaneous refinements of places in communicating components with communicating subcomponents.

In [12], the authors introduce a notion of place refinement where a place is refined by a workflow net. We define this refinement operation on open Petri nets, and refine internal places by communicating OWNs.

The choice for OWNs with a single initial and a single final place makes the refinement definition natural. The refinement is only defined for the nets that do not overlap, except for (possibly) interface places. In some situations we will additionally require that input places of A may not be output of B or vice-versa, to prevent breaking the composability of A with other components. These requirements are captured in the following definition.

Definition 22 (Refinable). *Let A be an OPN, and let B be an OWN. A* is refinable by *B if $(P_A \cup I_A \cup O_A \cup T_A) \cap (P_B \cup I_B \cup O_B \cup T_B) = (I_A \cup O_A) \cap (I_B \cup O_B)$.*

A is strictly refinable by *B if $(P_A \cup I_A \cup O_A \cup T_A) \cap (P_B \cup I_B \cup O_B \cup T_B) = (I_A \cap I_B) \cup (O_A \cap O_B)$.*

When we refine an arbitrary place $p \in P_A$ by an OWN B, all transitions of the refined net that produced a token in p now produce a token in the initial place of B, and all transitions that consumed a token from p now consume a token from the final place of B. If the two nets share interface places, these places are fused.

Definition 23 (Place refinement). *Let A be an OPN, and let B be an OWN such that A is refinable by B. Let $p \in P_A$ such that $i_A(p) = f_A(p) = 0$. The refined OPN $A \odot_p B = (P, I, O, T, F, i, f)$ is defined as:*

- $P = (P_A \setminus \{p\}) \cup P_B \cup (I_A \cap O_B) \cup (I_B \cap O_A)$;
- $I = (I_A \setminus O_B) \cup (I_B \setminus O_A)$;
- $O = (O_A \setminus I_B) \cup (O_B \setminus I_A)$;
- $T = T_A \cup T_B$;

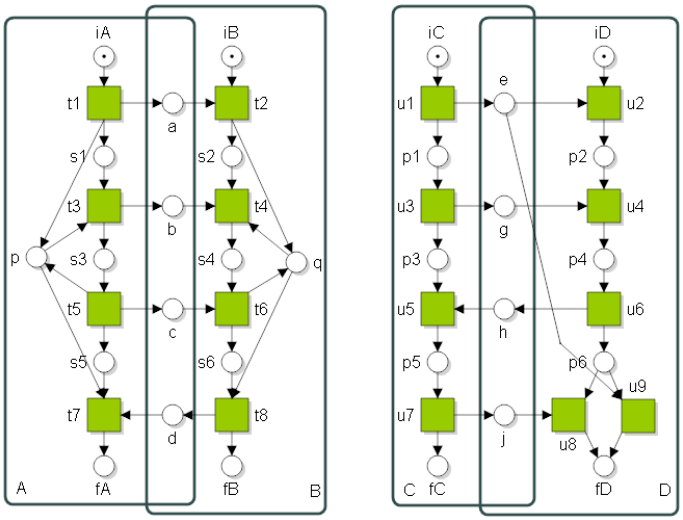

Fig. 2. Places p and q are safe, $A \oplus B \succeq A$ and $C \oplus D \succeq C$, but $(A \odot_p C) \oplus (B \odot_q D)$ is not sound

- $F = F_A \setminus ((^\bullet p \times \{p\}) \cup (\{p\} \times p^\bullet)) \cup F_B \cup (^\bullet p \times \{i_B\}) \cup (\{f_B\} \times p^\bullet)$;
- $i = i_A$;
- $f = f_A$.

Note that in case an input place of one net is an output place of another net, this place becomes an internal place of the resulting net.

If in the original net two places are refined, the resulting net does not depend on the order in which the refinements took place. Also, the refinement distributes over the composition.

Lemma 24. *Let N be an OPN, let $p, q \in P_N$ and $p \neq q$. Let C and D be two disjoint OWNs such that N is strictly refinable by C and N is strictly refinable by D. Then $(N \odot_p C) \odot_q D = (N \odot_q D) \odot_p C$. Furthermore, if $N = A \oplus B$ for some OPNs A and B, $p \in P_A$ and $q \in P_B$, we have $((A \oplus B) \odot_p C) \odot_q D = (A \odot_p C) \oplus (B \odot_q D)$.*

The following statement is a generalization of Theorem 10 from [12] to the case of OPN nets.

Lemma 25 (Soundness of refinement). *Let A be a safe and sound OPN, and let B be a sound OWN such that A is strictly refinable by B. Let $p \in P_A$. Then the refinement $A \odot_p B$ is sound.*

When constructing services, we might want to refine two places of communicating components with a sound composition of two subcomponents. Refinement

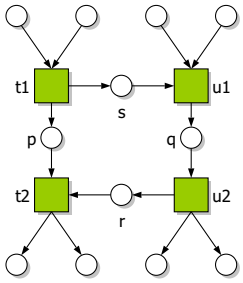

Fig. 3. Block structure ensuring that p is synchronized with q ($p \rightleftharpoons_N q$)

of a sound and safe OPN by a sound OWN results in a sound OPN, but refining two places in a sound and safe OPN by a sound composition is in general not sound. An intuitive approach would be to apply this refinement only to "synchronizable" places — such places that if one of the places becomes marked in the execution sequence, then another one already has a token, or will receive it before the token disappears from the first one. A counterexample for this idea is given in Figure 2. Both compositions $A \oplus B$ and $C \oplus D$ are safe and sound and places p and q are "synchronizable", but the composition $(A \odot_p C) \oplus (B \odot_q D)$, i.e. place p is refined by C and place q by D, is not sound, which is caused by the fact that C can be *started for the second time before D has finished*. Consider for example the firing sequence $\sigma = \langle t1, u1, u3, t2, u2, u4, u6, u5, u7, t3, t5, u1 \rangle$. We have $(A \odot_p C) \oplus (B \odot_q D) : [i_A, i_B] \xrightarrow{\sigma} [s5, b, c, s2, p1, e, j, p6]$. Continuing with the firing sequence $\gamma = \langle u9, u3, t4, t6 \rangle$ we obtain marking $[s5, s6, p3, g, j, i_D]$, which is a deadlock. This scenario was not possible in $C \oplus D$ itself.

To solve this problem, we need to ensure a stricter synchronization for places p and q, namely that place q only becomes marked if p is marked, and that p can only become unmarked, after q became unmarked. A structure that guarantees this condition is the *block structure* (see Figure 5), which is a handshaking protocol providing the notifications when a place becomes marked or unmarked. If place p becomes marked, this is notified by sending a message. This message is consumed when q becomes marked. As soon as q becomes unmarked, a message is sent, after which p can become unmarked.

Definition 26 (Synchronized places, $p \rightleftharpoons_N q$). *Let N be an OPN and $p, q \in P_N$ two distinct places. We say that p is* synchronized *with q in N, denoted by $p \rightleftharpoons_N q$, if and only if p and q are safe, $i_N(p) = i_N(q) = f_N(p) = f_N(q) = 0$, and there exist two* communication places *$r, s \in P_N$ and four transitions $t_1, t_2, u_1, u_2 \in T_N$ that together form a so-called* block structure *where:*

- ${}^\bullet p = \{t_1\} \wedge p^\bullet = \{t_2\} \wedge {}^\bullet q = \{u_1\} \wedge q^\bullet = \{u_2\}$;
- ${t_1}^\bullet = \{p, s\} \wedge {}^\bullet t_2 = \{p, r\} \wedge {u_1}^\bullet = \{q\} \wedge {}^\bullet u_2 = \{q\}$;
- ${}^\bullet s = \{t_1\} \wedge s^\bullet = \{u_1\} \wedge {}^\bullet r = \{u_2\} \wedge r^\bullet = \{t_2\}$.

The block structure guarantees that if place p becomes marked, it cannot become unmarked before q has been marked. This is ensured via two communication places, place s and place r. A token in place s indicates that a token is put in p but not yet in q, whereas a token in place r indicates that a token has

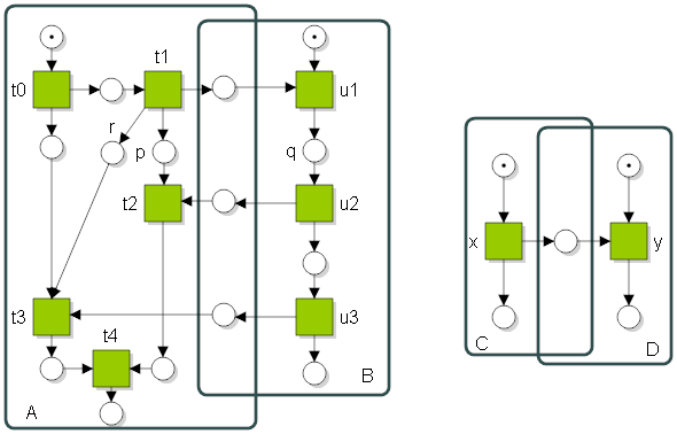

Fig. 4. Places p and q are not synchronized

been consumed from place q. I.e., a transition that produces a token in p should also produce a token in place s, a transition that consumes from s should place a token in q. If a transition consumes from place q, it should produce a token in r, which can only be consumed by a transition that also consumes from p. Note that the block structure is asymmetric, although reversing the direction of the messages will have small local influence on the behaviour when p and q get refined with communicating subcomponents.

It is easy to show that $p = q + r + s$ is a place invariant, i.e., the structure guarantees that the number of tokens in p equals the number of tokens in q, r, and s.

Lemma 27. *Let N be an OPN. Let $p, q \in P_N$ such that $p \rightleftharpoons_N q$. Let $s, r \in P_N$ be the communication places of the block structure of p and q. Then $p = q + s + r$ is a place invariant.*

This invariant implies that if the net is sound, the block structure of two synchronized places indeed guarantees the desired property: if p is marked, q is already marked, or it will become marked before the token is gone from p.

Corollary 28. *Let N be a sound OPN, $p, q \in P_N$ such that $p \rightleftharpoons_N q$ and $m \in \mathcal{R}(N)$ such that $m(p) = 1$. Then there exist a firing sequence $\sigma \in (T_N)^*$ and a marking $m' \in \mathcal{R}(N)$ such that $N : m \xrightarrow{\sigma} m'$ and $m'(p) = m'(r) = 1$ (i.e. σ enables t_2). Moreover, if $N = A \oplus B$ for some composable OPNs A and B with $A \oplus B \succeq A$, $p \in P_A$ and $q \in P_B$, then there exists such a sequence σ with just transitions of B (i.e., $\sigma \in (T_B)^*$).*

In Figure 4, we consider a slight extension of the block structure by allowing more output places for transition t_1. This extension does not work. First note that A, B and $A \oplus B$ are all sound and $A \oplus B \succeq A$. However, if we refine places

p and q with the very simple net $C \oplus D$, we loose the simulation property for the whole system. In $A \odot_p C$ we can have the firing sequence $\langle t_0, t_1, t_3 \rangle$ while in the whole system, $(A \odot_p C) \oplus (B \odot_q D)$, this firing sequence cannot be simulated, since the firing of transition x is needed in order to put a token on the third input place of t_3 in $(A \odot_p C) \oplus (B \odot_q D)$.

As in an OPN only a safe place can be refined by a sound OWN, we show that if a net is safe, and we refine two places by a safe composition of two OWNs, then the refined net is safe again.

Theorem 29 (Refinement preserves safety). *Let N be a safe OPN, let $p, q \in P_N$ such that $p \rightleftharpoons_N q$. Let C and D be two composable OWNs such that N is strictly refinable by C and N is strictly refinable by D, and $C \oplus D$ is safe. Then the refinement $N' = (N \odot_p C) \odot_q D$ is safe.*

Proof. Since N is safe, p and q are never marked with two or more tokens. Hence, if $m(p) > 0$, transition $t_2 \in p^{\bullet_N}$ fires before transition $t_1 \in {}^{\bullet_N}p$ can fire again. The same holds for u_1 and u_2 because of the block structure. Therefore, $C \oplus D$ can never be restarted before it is finished. Hence, since both N and $C \oplus D$ are safe, N' is safe. □

We use Corollary 28 to show that if in a sound OPN N two synchronized places are refined by a sound composition, then the refined OPN is sound as well.

Theorem 30 (Refinement preserves soundness). *Let N be a sound OPN with $p, q \in P_N$ such that $p \rightleftharpoons_N q$. Let C and D be composable OWNs such that $C \oplus D$ is sound and N strictly refinable by C and N strictly refinable by D. Then $N' = (N \odot_p C) \odot_q D$ is sound.*

Proof. (Idea) We map an arbitrary token marking $m' \in \mathcal{R}(N')$ to the marking m of N by putting a token on p when C is marked, a token on q when D is marked and keeping the rest of the marking as is; note that m is reachable in N. Due to the soundness of N, $m \xrightarrow{\sigma} f_N$ for some $\sigma \in T_N^*$. Then we use the fact that p and q are safe and synchronized, and $C \oplus D$ is sound to fill in σ up to σ' such that $m' \xrightarrow{\sigma'} f_{N'}$. □

The next theorem exploits the composition structure. If $A \oplus B$ simulates A and $C \oplus D$ simulates C, then $((A \odot_p C) \oplus (B \odot_q D))$ simulates $A \odot_p C$.

Theorem 31 (Refinement preserves simulation). *Let A and B be two composable OPNs such that $N = A \oplus B$, $N \succeq A$, and A is sound. Let $p \in P_A$ and $q \in P_B$ such that $p \rightleftharpoons_N q$. Let C and D be two composable OWNs such that N is strictly refinable by C and N is strictly refinable by D, $C \oplus D \succeq C$ and C is sound. Let $A' = A \odot_p C$ and $B' = (B \odot_q D)$. Then $A' \oplus B' \succeq A'$.*

Proof. (Sketch) Let $N' = A' \oplus B'$. We prove that $R = \{(m_{|P_{A'}}, m) \mid m \in \mathcal{R}(N')\}$ is a simulation, assuming all transitions of B' are τ-labeled. Let $m, m' \in \mathcal{R}(A')$, $t \in T_{A'}$ and $\overline{m} \in \mathcal{R}(N')$. Then either $t \in T_A$ or $t \in T_C$. Suppose $t \in T_A$. Then

either (1) C and D do not contain any marked place in m, or (2) there is at least one place marked in C. In the first case, the firing of t does not depend on the marking in p. $A \oplus B \succeq A$ implies the existence of a marking $\overline{m}' \in \mathcal{R}(N')$ such that $N' \colon \overline{m} \stackrel{t}{\Longrightarrow} \overline{m}'$ and $m' \, R \, \overline{m}'$. In the second case, we need to consider two subcases: either p is in the preset of t in A, or not. If p is not in the preset, the argument is similar to case (1). In case p is in the preset of t, we have $t = t_2$. If there is a place marked in D, $C \oplus D \succeq C$ implies that a marking $\overline{m}'' \in \mathcal{R}(N')$ enabling transition t can by reached by the firings of transitions of B and D only. Hence, there exists a marking $\overline{m}' \in \mathcal{R}(N')$ such that $N' : \overline{m} \stackrel{t}{\Longrightarrow} \overline{m}'$ and $m' \, R \, \overline{m}'$. If there is no place marked in D, then by Corollary 28, either t is enabled in $\overline{m}'$, or there exists a firing sequence in B marking a place in D. In both cases we can conclude the existence of a marking $\overline{m}' \in \mathcal{R}(N')$ such that $N' : \overline{m} \stackrel{t}{\Longrightarrow} \overline{m}'$ and $m' \, R \, \overline{m}'$.

Suppose $t \in T_C$. Then either (1) there exists a place in D that is marked in $\overline{m}$ or (2) no place in D is marked in $\overline{m}$. In the first case, $C \oplus D \succeq C$ implies the existence of a marking $\overline{m}'$ such that $N' : \overline{m} \stackrel{t}{\Longrightarrow} \overline{m}'$ and $m' \, R \, \overline{m}'$. If there is no place in D that is marked in $\overline{m}$, then by Corollary 28, either there is a firing sequence in B that marks a place in D or D has already finished. In the first case, a marking $\overline{m}'' \in \mathcal{R}(N')$ with $N' \overline{m} \Longrightarrow \overline{m}''$ is reached, marking a place in D, hence we can apply case (1) to marking $\overline{m}''$. In the second case, D has already produced all necessary tokens for C, thus $t \leq \overline{m}$. In both cases we can conclude the existence of a marking $\overline{m}' \in \mathcal{R}(N')$ such that $N' : \overline{m} \stackrel{t}{\Longrightarrow} \overline{m}'$ and $m' \, R \, \overline{m}'$. □

Theorems 21 and 31 imply compositionality of our construction method with respect to soundness:

Corollary 32. *Let A and B be two composable OPNs such that $N = A \oplus B$, $N \succeq A$, and A sound. Let $p \in P_A$ and $q \in P_B$ such that $p \rightleftharpoons_N q$. Let C and D be two composable OWNs such that N is strictly refinable by C and N is strictly refinable by D, $C \oplus D \succeq C$, and C sound. Then $((A \odot_p C) \oplus (B \odot_q D))$ is sound.*

6 Construction of Service Trees

In Section 4 we defined a sufficient condition for the soundness of service trees and showed that this condition is equivalent to the simulation property. In Section 5, we showed that if two places in a composition are synchronized, they can be refined by a composition, such that the refined net is sound again. In this section, we combine the results obtained so far to construct service trees in a soundness-by-construction fashion. We show two examples of possible approaches, one for OWNs and another one for a specific subclass of OPNs.

The first approach is inspired by the concept of outsourcing. Consider a sound OWN N. Suppose some place $x \in P_N$ is not just a state marker, but it represents the execution of an activity outside the scope of the OWN, e.g. the place is called "producing an item". Now suppose there is a service that produces this "item".

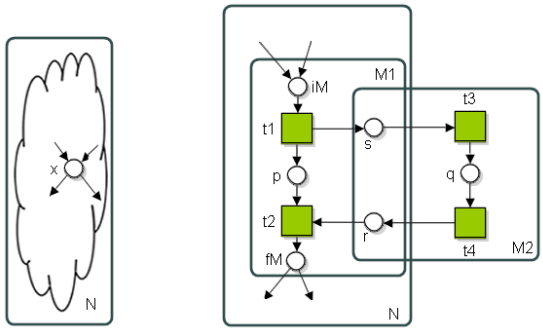

Fig. 5. The "outsourcing" method: place x is refined by M_1

Then we can "outsource" the activity to the service we found. We modify the net, and refine the place by a start transition and an end transition, indicating the start and end of the activity. The start transition initiates the service, and as soon as the service finishes, the end transition is triggered.

Consider Figure 5, where place x represents "producing an item" and we assume that it is safe. By Lemma 25, we can refine place x by a sound workflow net so that the refined net is sound. Consider the OWN $M_1 = (\{i_M, p, f_M\}, \{r\}, \{s\}, \{t_1, t_2\}, \{(i_M, t_1), (t_1, p), (t_1, s), (p, t_2), (r, t_2), (t_2, f_M)\}, [i_M], [f_M])$. The refinement $N \odot_x M_1$ is sound, and, since place x is safe, place p is safe. Now, consider the OPN $M_2 = (\{q\}, \{s\}, \{r\}, \{t_3, t_4\}, \{(s, t_3), (t_3, q), (q, t_4), (t_4, r)\}, \emptyset, \emptyset)$. Although we need to drop the requirement that the final marking is a deadlock, it is easy to show that the net $(N \odot_x M_1) \oplus M_2$ is sound and so is $(N \odot_x M_1) \oplus M_2 \succeq (N \odot_x M_1)$. By this composition, we introduced a block structure around places p and q (see Figure 5). By Lemma 27 and the safety of p, we know that q is safe, and thus $p \rightleftharpoons_{(N \odot_x M_1) \oplus M_2} q$. Now we can refine places p and q by any sound composition of OWNs. This way, we can construct an arbitrary large sound service tree, without the need to check condition $\Omega_{A,B}$ for any pair of composed OPNs A and B.

Theorem 33. *Let N be an arbitrary OPN and $x \in P_N$. Consider the two OPNs M_1 and M_2 from Figure 5. Let $N' = N \odot_x (M_1 \oplus M_2) = (N \odot_x M_1) \oplus M_2$. Then:*

1. *if N is safe, then N' is safe;*
2. *if N is sound, then N' is sound;*
3. *$N' \succeq (N \odot_x M_1)$;*
4. *if N is an OWN, then N' is also an OWN.*

Proof. This follows immediately from the structure of M_1 and M_2. □

A second approach is based on pairs of composed OPNs belonging to a special subclass $\mathcal{N}$ of OPNs. This class is recursively defined, starting with two composed OPNs that are either both acyclic T-nets[3] or both isomorphic S-nets[4] with some

[3] A Petri net in which all places have maximal one input and output transition
[4] A Petri net in which all transitions have maximal one input and one output place

additional requirements on their composition. The class $\mathcal{N}$ is defined in such a way that any pair of OPNs in this class is safe, sound and has the simulation property. In these pairs of composed OPNs we can easily detect the synchronized places and then each pair of these can be refined with nets of $\mathcal{N}$, resulting in an element that is again in $\mathcal{N}$. For details, see [4].

As an example, consider the approach presented in [13]. All services have the same protocol (OPN) for bargaining about the outsourcing of a service. Internally each service has its OWN orchestration process of the service. A task of this orchestration is modeled by a start transition, an end transition and a place in between like in Figure 5. The whole orchestration is modeled as an OWN. Some tasks may be outsourced to another service. The next step is refinement of the synchronized places by the standard bargaining protocol, which is in fact a sound composition of two OWNs, one for the service client and one for the service provider. In the OWN of the service provider there is one place that represents the execution of the service. This place may be refined by an arbitrary sound OWN which represents the orchestration of the service and by Lemma 25 this conserves the soundness of the whole system. Now we may select a task in this orchestration process and we may repeat the outsourcing process. Hence, we build up a service tree as an OWN.

Another example is that we have three parties with interaction between all three. For instance a Buyer invokes a service at a Seller and the Seller invokes a service at a Shipper which is the direct delivery of the goods to the Buyer. Now, the question is: "does this fit into the framework?". In fact the Seller process will wait in some place q until the Shipper has delivered the goods at the Buyer and then the Shipper will notify the Seller by sending a token to a transition t in the Seller, where t is an output transition of q. We may refine q by a OWN that only intercepts and passes the communication between Shipper and Buyer and thus all interaction between the Shipper and the Buyer is now via the Seller. This reflects the responsibility: the Seller is in fact responsible for the delivery. Hence, now it is a tree structure again and it fits into our framework.

7 Conclusions

In this paper, we presented a method for compositional verification of a global termination property of an arbitrary tree of communicating components, where only checks of pairs of directly linked components are required. Another dimension where our construction goes is place refinements. Note that our method can easily be extended to the simultaneous refinement of several (more than two) places in communicating components with communicating subcomponents. Finally, we gave a method to construct such a tree in a correctness-by construction fashion, based on the composition and refinement.

Related Work In [10] the authors give a constructive method preserving the inheritance of behavior. As shown in [2] this can be used to guarantee the correctness of interorganizational processes. Other formalisms, like I/O automata [17]

or interface automata [7] use synchronous communication, whereas we focus on asynchronous communication, which is essential for our application domain, since the communication in SOA is asynchronous.

In [20], the author introduces place composition to model asynchronous communication focusing on the question which subnets can be exchanged such that the behavior of the whole net is preserved. Open Petri Nets are very similar to the concept of Petri net components, see e.g. [14], in which a module consists of a Petri net and interface places to be connected to other components. Open Petri nets were introduced and studied in [5, 6, 15, 16, 18]. In [16] the authors focus on deciding *controllability* of an OPN and computing its *operating guidelines*. Operating guidelines can be used to decide substitutability of services [19], or to prove that an implementation of a service meets its specification [6].

The major advantage of our approach compared to the operating guideline approach is its compositionality. In our setting it is sufficient to analyze only directly connected services of the tree, while the overall operating guidelines of the tree would have to be re-computed before a new service can be checked for a harmless addition to the tree. Moreover, the construction of the service tree remains flexible — any component can be replaced by another component, provided that the composition of this component with its direct neighbors satisfies our condition. This flexibility comes however with a price label, namely, the condition we define is a sufficient but not necessary condition, i.e. we might not be able to approve some service trees, although they were sound.

In [11], the authors propose to model choreographies using Interaction Petri nets, which is a special class of Petri nets, where transitions are labeled with the source and target component, and the message type being sent. To check whether the composition is functioning correctly, the whole network of components needs to be checked, whereas in our approach the check is done compositionally.

Future Work The sufficient condition provided in Section 4 requires that an OPN B does not restrict the behavior of A in the composition $A \oplus B$. This condition might be relaxed by requiring that $A \oplus B$ mimics all *visible* behavior. The main research question here is defining a set of visible actions so that the approach would remain compositional and this set would be as small as possible. Note that such a relaxation will not influence the framework.

In Section 6, we have shown how the obtained results can be used to build service trees that are sound by construction. Although we only apply our results on Petri nets, our method can be extended to languages like BPEL, to facilitate the construction of web services in development environments like Oracle BPEL or IBM Websphere.

References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.
2. W.M.P. van der Aalst. Inheritance of Interorganizational Workflows: How to agree to disagree without loosing control? *Information Technology and Management Journal*, 4(4):345–389, 2003.

3. W.M.P. van der Aalst, M. Beisiegel, K.M. van Hee, D. König, and C. Stahl. An SOA-Based Architecture Framework. *International Journal of Business Process Integration and Management*, 2(2):91–101, 2007.
4. W.M.P. van der Aalst, K.M. van Hee, P. Massuthe, N. Sidorova, and J.M.E.M. van der Werf. Compositional service trees. Technical Report CSR 09/01, Technische Universiteit Eindhoven, January 2009.
5. W.M.P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf. From Public Views to Private Views: Correctness-by-Design for Services. In *WS-FM 2007*, volume 4937 of *Lecture Notes in Computer Science*, pages 139–153. Springer, 2008.
6. W.M.P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf. Multiparty Contracts: Agreeing and Implementing Interorganizational Processes. *The Computer Journal*, 2009. (Accepted for publication).
7. L. de Alfaro and Th. A. Henzinger. Interface automata. *SIGSOFT Softw. Eng. Notes*, 26(5):109–120, 2001.
8. G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services Concepts, Architectures and Applications*. Springer, 2004.
9. A. Alves, A. Arkin, S. Askary, et al. Web Services Business Process Execution Language Version 2.0 (OASIS Standard). WS-BPEL TC OASIS, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, 2007.
10. T. Basten and W.M.P. van der Aalst. Inheritance of Behavior. *Journal of Logic and Algebraic Programming*, 47(2):47–145, 2001.
11. G. Decker and M. Weske. Local enforceability in interaction petri nets. In *BPM 2007*, volume 4714 of *Lecture Notes in Computer Science*, pages 305–319. Springer, 2007.
12. K.M. van Hee, N. Sidorova, and M. Voorhoeve. Soundness and Separability of Workflow Nets in the Stepwise Refinement Approach. In *ICATPN 2003*, volume 2679 of *LNCS*, pages 335–354. Springer, 2003.
13. K.M. van Hee, H.M.W. Verbeek, C. Stahl, and N. Sidorova. A framework for linking and pricing no-cure-no-pay services. *Transactions on Petri Nets and Other Models of Concurrency*, 2009. to appear.
14. E. Kindler. A compositional partial order semantics for Petri net components. In *18th International Conference on Application and Theory of Petri Nets (ICATPN 1997)*, volume 1248 of *LNCS*, pages 235–252. springer, June 1997.
15. N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg. Analyzing Interacting BPEL Processes. In *BPM 2006*, volume 4102 of *Lecture Notes in Computer Science*, pages 17–32. Springer, 2006.
16. N. Lohmann, P. Massuthe, and K. Wolf. Operating Guidelines for Finite-State Services. In *ICATPN 2007*, volume 4546 of *Lecture Notes in Computer Science*, pages 321–341, Siedlce, Poland, jun 2007. Springer.
17. N. A. Lynch and M. Tuttle. Hierarchical correctness proofs for distributed algorithms. In *6th Annual ACM Symposium on Principles of Distributed Computing*, 1987.
18. P. Massuthe, W. Reisig, and K. Schmidt. An Operating Guideline Approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics*, 1(3):35–43, 2005.
19. C. Stahl, P. Massuthe, and J. Bretschneider. Deciding Substitutability of Services with Operating Guidelines. *Transactions on Petri Nets and Other Models of Concurrency*, 2008. (Accepted for publication).
20. W. Vogler. Asynchronous communication of petri nets and the refinement of transitions. In *Automata, Languages and Programming*, volume 623 of *LNCS*, pages 605 – 616. springer, 1992.