Compositional Process Model Synthesis Based on Interface Patterns

Roman A. Nesterov(✉) and Irina A. Lomazova

National Research University Higher School of Economics,
20 Myasnitskaya Ulitsa, 101000 Moscow, Russia
ranesterov@edu.hse.ru, ilomazova@hse.ru

Abstract. Coordination of several distributed system components is an error-prone task, since interaction of several simple components can generate rather sophisticated behavior. Verification of such systems is very difficult or even impossible because of the so-called state space explosion problem, when the size of the system reachability set grows exponentially on the number of interacting agents. To overcome this problem several approaches to construct correct models of interacting agents in a compositional way were proposed in the literature. They define different properties and conditions to ensure correct behavior of interacting agents. Checking these conditions may be in its turn quite a problem.

In this paper we propose patterns for correct composition of component models. For justifying these patterns we use special net morphisms. However, to apply patterns the user does not need to be familiar with the underlying theory.

Keywords: Petri nets · Distributed systems · Interface patterns Synchronization · Compositionality · Morphisms

1 Introduction

The development of correct distributed systems meets several hard problems. One of them is to define correct coordination of various system components meeting a specification of their interaction. This task is rather complicated and error-prone, since bad organization of interaction can lead to deadlocks, or violate proper termination of component behavior.

On the other hand, distributed system models can be too large and complicate for verification with the existing tools. The solution is to develop techniques for compositional building system models from component models, in the way which guarantees correctness of system behavior.

Here we study the following problem of defining component interaction: given two deadlock-free and properly terminating component models and a scheme of

This work is supported by the Basic Research Program at the National Research University Higher School of Economics and Russian Foundation for Basic Research, project No. 16-01-00546.

their interaction we need to construct the deadlock-free and properly terminating system model. We will compose interacting components by applying special patterns specifying different ways of interaction. In this paper we define several patterns for constructing a correct system model from correct component models. The correctness of patterns is justified with the help of special net morphisms, defined in [3]. However, to apply the patterns it is not needed to know the underlying theory.

Thus, our aim is to define compositional patterns for typical interaction schemes, which can be used by the developer to construct reliable models.

2 Related Work and Motivation

Petri nets [13] is one of the most popular formalisms for modeling and analysis of distributed systems. We use Petri nets for representing component models as well as interface scheme. Then component models should be composed into one Petri net model, which represents the overall system behavior according to the interface definition.

Petri net composition was extensively studied in the literature. Petri nets can be composed via transition merging, which corresponds to synchronization of events, or via place merging corresponding to asynchronous communication by resource sharing. The general framework of such composition is described in [13]. Petri net composition by place/transition merging is intuitively clear and easy to implement. The problem with it is that the resulting system does not inherit components behavioral properties, since after composing with some other components the component behavior can be crucially changed. So, after such composition the model should be verified from scratch.

One of possible ways to achieve inheritance of component behavioral properties is to use net morphisms [2,14]. Special constructs for composing Petri net components based on net morphisms were studied in [3–5]. The key idea of this approach is that distributed system components refine an abstract interface, which describes the interaction between them. The composition based on morphisms provides support for the modular system development process [1,12].

A rather large number of studies have been devoted to the problem of compositional web service synthesis using different classes of Petri nets [6–9,15,16]. These works define and study different kinds of templates for composing services, but they do not consider the restriction on execution order among services. In the overview [6] the authors stress that there is a lack in execution engines or frameworks based on Petri Nets.

In their early work [8] R. Hamadi and B. Benatallah have offered a systematic algebraic approach to regular composition of services via sequencing, choice, concurrency etc. (see Fig. 1). Applying these operations to proper terminating component models gives a model, which proper terminates by construction.

The composition operations in [8] ignore inner structure of the composed components, there is no possibility to specify e.g. the order of inner actions in two components. This case is schematically represented in Fig. 2. Here the first

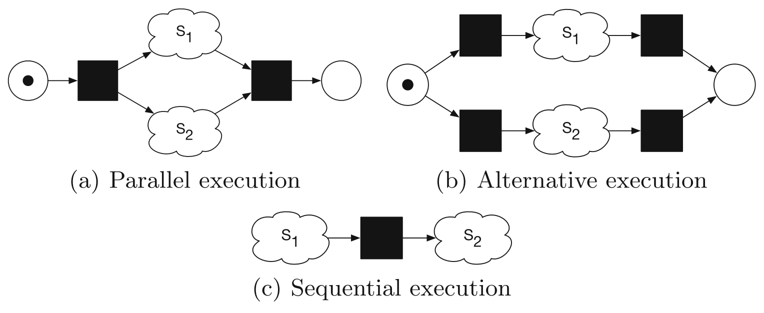


Fig. 1. Templates for composing two web services S_1 and S_2

component includes an inner action **A**, which occurs in all component executions. The second component has an inner action **B** with the same property. The interaction scheme requires that **A** should be implemented before **B** (e.g. **B** uses a resource produced by **A**). Then the problem is to define a pattern for composing two components via given interface in such a way that the target model inherits proper termination of both components.

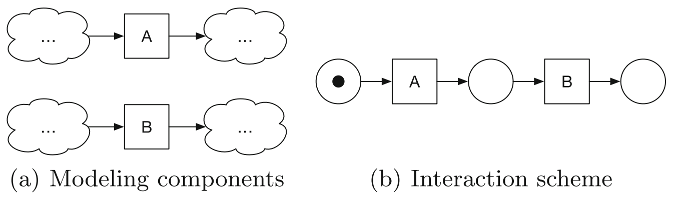


Fig. 2. Extending templates to relations on inner actions of components

This paper gives a solution to this and two other typical interaction patterns. We use special net morphisms to justify correctness of the obtained models.

3 Composing Petri Nets via Interface Patterns

We start this section by giving background definitions concerning Petri nets and their behavior.

Definition 1. *A Petri net is a bipartite graph $N = (P, T, F, m_0, L)$, where:*

1. *$P = \{p_1, p_2, p_3, ..., p_n\}$ – a finite non-empty set of places.*
2. *$T = \{t_1, t_2, t_3, ..., t_m\}$ – a finite non-empty set of transitions, $P \cup T = \varnothing$.*
3. *$F \subseteq (P \times T) \cup (T \times P)$ – a flow relation.*
4. *$m_0 \subseteq P$ – an initial marking (configuration) of a net N.*
5. *$L : T \to \{\mathcal{A}\} \cup \tau$ – a labeling for transitions, where τ is a name for **silent** transitions.*

Let t be a transition in a Petri net N. We call a set ${}^{\bullet}t = \{p \in P | (p,t) \in F\}$ a **preset** of t and a set $t^{\bullet} = \{p \in P | (t,p) \in F\}$ – a **postset** of t. Subsequently, ${}^{\bullet}t^{\bullet} = {}^{\bullet}t \cup t^{\bullet}$ is called a **neighborhood** of t.

The behavior of Petri nets is defined through the firing rule, which specifies when an action can occur, and how it modifies the overall state of a system.

A marking $m \subseteq P$ **enables** a transition t, denoted $m[t\rangle$, if ${}^{\bullet}t \subseteq m$ and $t^{\bullet} \cap m = \varnothing$. The t firing in m leads to m', denoted $m[t\rangle m'$, where $m' = m \setminus {}^{\bullet}t \cup t^{\bullet}$. When $\forall t \in T$ and $\forall w \in T^{*}$, $m[tw\rangle m' = m[t\rangle m''[w\rangle m'$, w is called the **firing sequence**. We denote a set of all firing sequences of a net N as $FS(N)$.

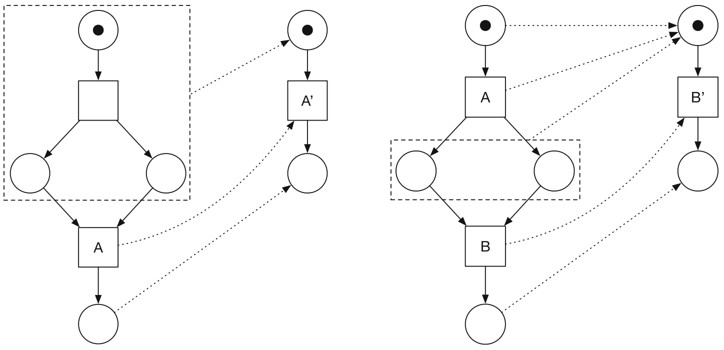
We call a marking $m \subseteq P$ **reachable** if $\exists w \in FS(N) : m_0[w\rangle m$. A set of all reachable markings of a net N is $[m_0\rangle$. A reachable marking m is **dead** if it does not enable any transition. A marking $m \subseteq P$ is called **final** if $\forall p \in m : p^{\bullet} = \varnothing$. A net N is **deadlock-free** if $\forall t \in T \exists m \in [m_0\rangle : m[t\rangle$ except m is a final marking. A net N **terminates properly** if a final marking is reachable.

The challenge of this work is to compose models of separate components into a single system model preserving crucial properties of the initial models. Our approach is based on the notion of ω-morphisms introduced in [3].

Definition 2. *Let $N_i = (P_i, T_i, F_i, m_0^i)$ for $i = 1, 2$ be two acyclic Petri nets, $X_i = P_i \cup T_i$. The ω-morphism is a **total** surjective map $\varphi : X_1 \to X_2$ such that:*

1. *$\varphi(P_1) = P_2$, $\varphi(m_0^1) = m_0^2$.*
2. *$\forall t_1 \in T_1$, if $\varphi(t_1) \in T_2$, then $\varphi({}^{\bullet}t_1) = {}^{\bullet}\varphi(t_1)$ and $\varphi(t_1{}^{\bullet}) = \varphi(t_1)^{\bullet}$.*
3. *$\forall t_1 \in T_1$, if $\varphi(t_1) \in P_2$, then $\varphi({}^{\bullet}t_1{}^{\bullet}) = \{\varphi(t_1)\}$.*

Figure 3 explains the requirements 2 and 3 of this definition. To compose two component nets we need to define morphisms from them towards the abstract interface (system view) they refine. After morphisms are defined, we merge structural elements mapped onto the same interface places preserving flow relation of both components.



(a) $\varphi(A) = A' \in T_2$ (b) $\varphi(A) = p \in P_2$

Fig. 3. Transition map options for ω-morphisms

Another intention of our study is to provide patterns covering generic cases of distributed process modeling. We construct component nets from abstract subnets denoted $N_i(S_j)$ where N_i and S_j are a component and a subnet labels correspondingly. They are also represented in a form of Petri nets. It has to be noted that we explicitly identify a set of subnet input and output places such that $^\bullet p = \varnothing$ and $p^\bullet = \varnothing$. We use single places to represent these sets. Figure 4 shows how we depict this subnet and how we aim to combine them via transitions in order to represent a behavior of a system component. Our study considers deadlock-free and proper terminating subnets.

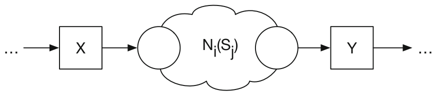


Fig. 4. An abstract subnet

Prior to composing two Petri nets via ω-morphisms we need to obtain so-called canonical representations of them with respect to an interface net.

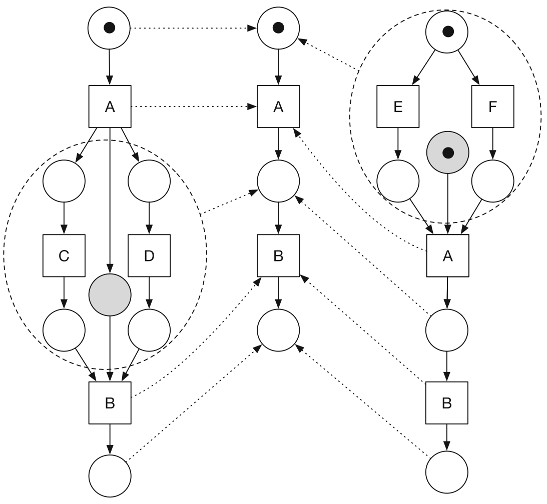
Definition 3. *Let $N_i = (P_i, T_i, F_i, m_0^i, L_i)$ be two Petri nets for $i = 1, 2$. A canonical representation of N_1 with respect to N_2, denoted $N_1^C = (P, T, F, m_0, L)$ is defined as follows:*

1. *$P = P_1 \cup P_2$ and $m_0 = m_0^1 \cup m_0^2$.*
2. *$T = T_1 \cup T_2$, identically labeled transitions are merged correspondingly.*
3. *$F = F_1 \cup F_2$.*
4. *$L : T \to \{\mathcal{A} \cup \tau\}$, $\forall t_1 \in T_1 : L(t_1) = L_1(t_1)$ and $\forall t_2 \in T_2 : L(t_2) = L_2(t_2)$.*

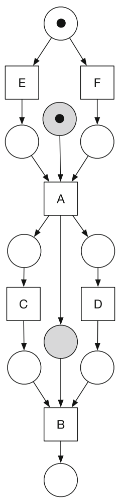
Afterwards we can compose two canonical representations of component Petri nets via defining ω-morphisms from them to another interface net specifying requirements for their interaction. Figure 5 explains how to obtain a composition according to the definition below.

Definition 4. *Let $N_i = (P_i, T_i, F_i, m_0^i, L_i)$ be Petri nets for i=1, 2, I. Let N_1^C, N_2^C be canonical representations of N_1 and N_2 with respect to N_I and $\omega_i : N_i \to N_I$ for $i = 1, 2$ be two ω-morphisms. A Petri net $N = (P, T, F, m_0, L)$ is called a composition of N_1 and N_2 via interface N_I and morphisms ω_1, ω_2 (denoted $N = N_1 \langle N_I \rangle N_2$) iff N is obtained from N_1, N_2, N_I by merging transitions with the same labels related by w_1 and w_2, i.e. transitions t and t' are merged iff $L(t) = L(t')$ and $\omega(t) = t'$.*

It can be easily seen that there are redundant places in a composition N, which do not influence its behavior (see Fig. 5(b)). We can reduce a composition by removing these places with the help of simple reduction rules proposed by T. Murata [10]. Further when we describe patterns, we provide already reduced composition of models.



(a) Morphisms ω_1 and ω_2



(b) N

Fig. 5. A composition of two Petri nets via ω-morphisms

The composed net is deadlock-free and terminates properly according to the following proposition based on [4] where the proof based on weak bisimilarity notion [11] is provided.

Proposition 1. *Let N_1 and N_2 be two Petri nets representing components, and N be their composition obtained via the interface Petri net N_I and ω-morphisms as described above.*

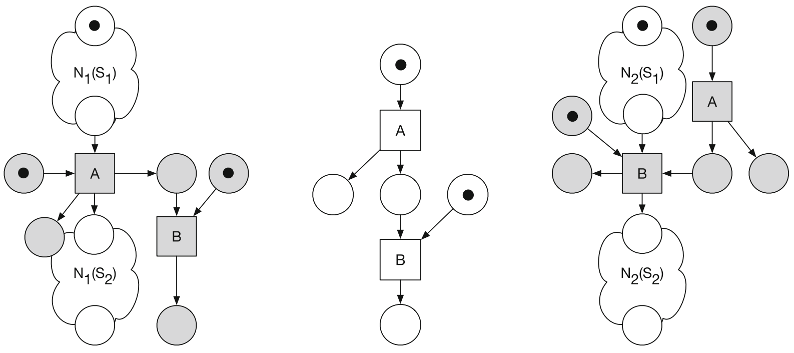
1. *The Petri net N is deadlock-free, if N_1, N_2 and N_I are deadlock-free.*
2. *The Petri net N terminates properly, if N_1, N_2 and N_I terminates properly.*

Finally, while describing patterns we also want to preserve mutual independence of components, i.e. a composition must not add any behavioral constraints that are not provided in the component nets.

However, the original interfaces we propose in problem statements for patterns sometimes are not appropriate for the preservation of mutual independence. To overcome this problem, we construct an extended interface net which performs the same sequences of observable actions. This equivalence is called string equivalence in [11]. This extension preserves component independence.

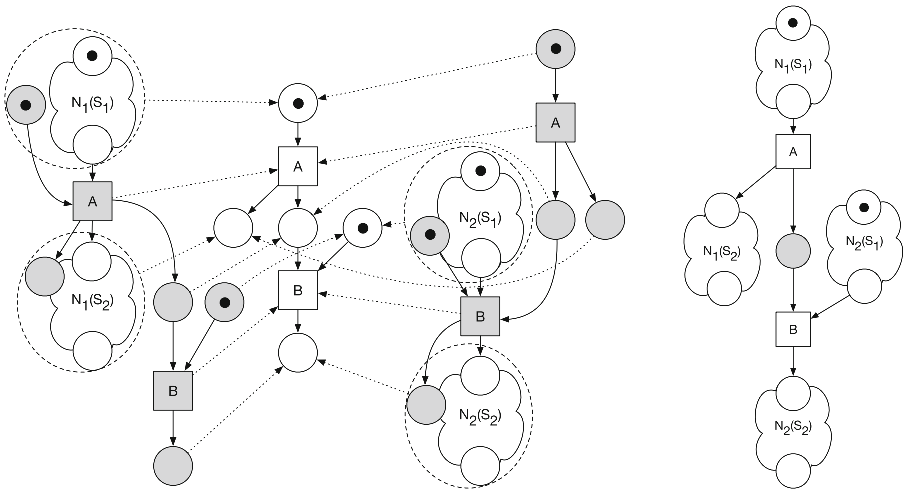
3.1 Pattern 1: Simple Causality

In this subsection we implement the composition in accordance with the simple causality pattern (see Fig. 2) and provide all explanatory details. The causality can mean, for instance, that in a composed net the first component while executing **A** produces the necessary resources for the correct implementation of **B** by the second component.



(a) N_1^C (b) Interface N_I (c) N_2^C

Fig. 6. Canonical component representations



(a) Morphisms ω_1, ω_2 (b) The composition

Fig. 7. The composition for the simple causality pattern via ω-morphisms

Firstly, we adjust the original interface net to preserve the mutual independence of the components by constructing a Petri net shown in Fig. 6(b). Figure 6(a) and (c) show the canonical representations of component nets with respect to the interface. Afterwards, we define ω-morphisms from the component nets towards the interface (see dotted arrows in Fig. 7(a)) and obtain composition shown in Fig. 7(b) after some simple reductions.

3.2 Pattern 2: Extended Causality

In this section we generalize the simple causality pattern by adding concurrent branches in component models. In this case both component models look as shown in Fig. 8. The interface here is identical to the interface we used for simple causality pattern (see Fig. 2). The extended version of the original interface is shown in Fig. 9.

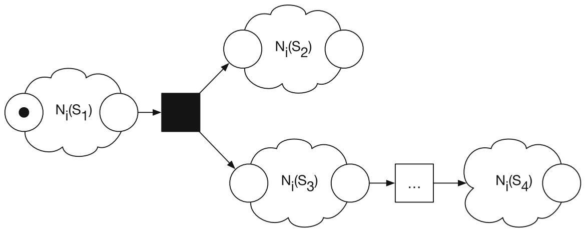


Fig. 8. The component net with concurrent branches

Then we define ω-morphisms and compose source component models. The reduced result is shown in Fig. 10. It can be easily seen, that we have got this net by connecting transitions **A** and **B** via additional control place similar to the case of the simple causality pattern.

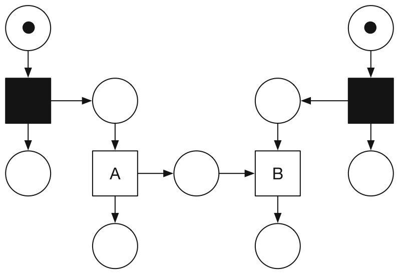


Fig. 9. The interface net for the extended causality

Morphisms help us find which transitions should be connected and and how to organize this connection. As a result, we obtain a correct (deadlock-free and proper terminating) composition by construction.

3.3 Pattern 3: Conditional Causality

Conditional causality is another generalization of the simple causality pattern by adding choice construct in one of the two interacting components. Figure 11 shows the problem statement for composing two components following this pattern. The main feature of the conditional causality is that we choose between two sequences of actions having one action in common that is **always** executed when these components interact.

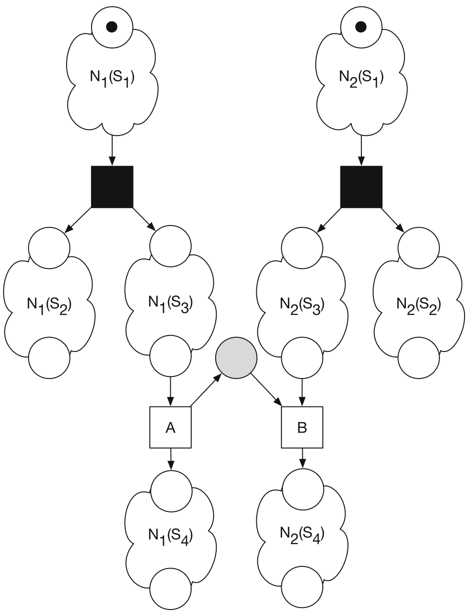


Fig. 10. The composition for the extended causality pattern via ω-morphisms

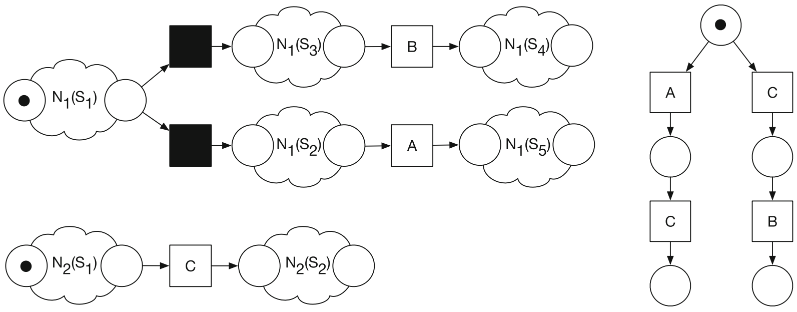


Fig. 11. The components and the interface for the conditional causality pattern

The interface shown in Fig. 11(b) cannot be directly used for defining morphisms. Two copies of the same action **C** in the interface net do not allow to define surjective maps. To overcome this problem we construct an interface net with a single copy of the action **C** as shown in Fig. 12. The idea behind this construction is quite straightforward: we need to remember what option we choose to execute (**A-C** or **C-B**) and that **C-B** sequence fires if **A** does not fire.

After defining ω-morphisms, composing nets, and making some reductions we get the Petri net the composition result for the conditional causality pattern

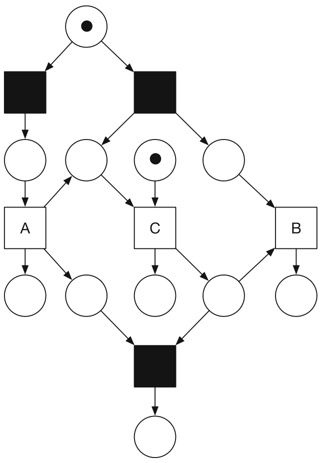


Fig. 12. The interface net for the conditional causality

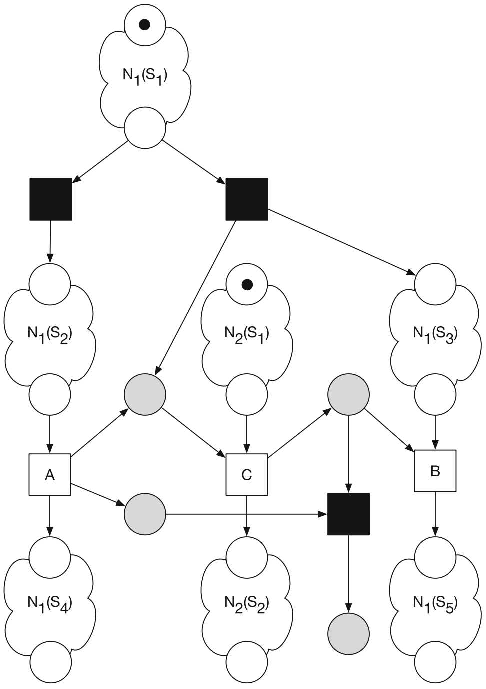


Fig. 13. The composition for the conditional causality pattern via ω-morphisms

is shown in Fig. 13. Apart from meeting the requirements imposed by the original interface net (see Fig. 11(b)), we get the composition with clearly identified components.

4 Conclusion

In this paper we have proposed an approach for constructing models of distributed systems in a compositional way. The key idea is to automatically obtain the correct and complete process models from the separate source models of its components.

The proposed approach allows us to compose Petri net models respecting some relations on inner actions of the components.

We have constructed three patterns for the composition of two interacting components. The suggested templates can be used for manual or automatic synthesis of models. The composed model is deadlock-free and terminates properly, provided all components satisfy these properties. It also clearly represents component models as parts of the target model.

The future research will be focused on developing patterns for other relations including such patterns as the exclusive choice and on combining several patterns together and defining patterns for more than two components.

References

1. Bednarczyk, M.A., Bernardinello, L., Caillaud, B., Pawłowski, W., Pomello, L.: Modular system development with pullbacks. In: van der Aalst, W.M.P., Best, E. (eds.) ICATPN 2003. LNCS, vol. 2679, pp. 140–160. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-44919-1_12
2. Bednarczyk, M.A., Borzyszkowski, A.M.: General morphisms of petri nets (Extended Abstract). In: Wiedermann, J., van Emde Boas, P., Nielsen, M. (eds.) ICALP 1999. LNCS, vol. 1644, pp. 190–199. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48523-6_16
3. Bernardinello, L., Mangioni, E., Pomello, L.: Local state refinement and composition of elementary net systems: an approach based on morphisms. In: Koutny, M., Aalst, W.M.P., Yakovlev, A. (eds.) Transactions on Petri Nets and Other Models of Concurrency VIII. LNCS, vol. 8100, pp. 48–70. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40465-8_3
4. Bernardinello, L., Monticelli, E., Pomello, L.: On preserving structural and behavioural properties by composing net systems on interfaces. Fundamenta Informaticae **80**(1–3), 31–47 (2007)
5. Bernardinello, L., Pomello, L., Scaccabarozzi, S.: Morphisms on marked graphs. In: Moldt, D., Rlke, H. (eds.) International Workshop on Petri Nets and Software Engineering (PNSE 2014). CEUR Workshop Proceedings, No. 1160, pp. 113–127. CEUR-WS.org (2014)
6. Cardinale, Y., El Haddad, J., Manouvrier, M., Rukoz, M.: Web service composition based on petri nets: review and contribution. In: Lacroix, Z., Ruckhaus, E., Vidal, M.-E. (eds.) RED 2012. LNCS, vol. 8194, pp. 83–122. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45263-5_5
7. Feng, X.-N., Liu, Q., Wang, Z.: A web service composition modeling and evaluation method used petri net. In: Shen, H.T., Li, J., Li, M., Ni, J., Wang, W. (eds.) APWeb 2006. LNCS, vol. 3842, pp. 905–911. Springer, Heidelberg (2006). https://doi.org/10.1007/11610496_125

8. Hamadi, R., Benatallah, B.: A petri net-based model for web service composition. In: Proceedings of the 14th Australasian Database Conference, vol. 17, pp. 191–200. Australian Computer Society, Inc. (2003)
9. Lomazova, I.A.: Interacting workflow nets for workflow process re-engineering. Fundamenta Informaticae **101**(1–2), 59–70 (2010)
10. Murata, T.: Petri nets: Properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
11. Pomello, L., Rozenberg, G., Simone, C.: A survey of equivalence notions for net based systems. In: Rozenberg, G. (ed.) Advances in Petri Nets 1992. LNCS, vol. 609, pp. 410–472. Springer, Heidelberg (1992). https://doi.org/10.1007/3-540-55610-9_180
12. Pomello, L., Bernardinello, L.: Formal tools for modular system development. In: Cortadella, J., Reisig, W. (eds.) ICATPN 2004. LNCS, vol. 3099, pp. 77–96. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-27793-4_5
13. Reisig, W.: Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies. Springer Publishing Company, Incorporated (2013)
14. Winskel, G.: Petri nets, morphisms and compositionality. In: Rozenberg, G. (ed.) APN 1985. LNCS, vol. 222, pp. 453–477. Springer, Heidelberg (1986). https://doi.org/10.1007/BFb0016226
15. Xu, K., Ma, B.: A petri net based execution engine for web service composition. In: Huang, Z., Liu, C., He, J., Huang, G. (eds.) WISE 2013. LNCS, vol. 8182, pp. 181–193. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54370-8_16
16. Zhang, Z.l., Hong, F., Xiao, H.j.: A colored petri net-based model for web service composition. J. Shanghai Univ.(Engl. Edition) **12**(4), 323–329 (2008)