Abstraction-Based Deadlock Analysis of Service-Oriented Systems with Recursive Petri Nets

Erik Jonas Hartnick[1,*,†], Mandy Weißbach[1,*,†]

[1]Institute of Computer Science, Martin-Luther-University Halle-Wittenberg, Von-Seckendorff-Platz 1, 06120 Halle (Saale), Germany

Abstract

Service-oriented systems involve synchronous, asynchronous, and recursive service calls. Existing research has identified limitations in Petri net-based approaches for deadlock detection, particularly in scenarios involving recursive calls. This extended abstract establishes a basis for evaluating the suitability of recursive Petri nets for modeling such interactions and for identifying deadlocks, with an emphasis on recursion-induced cases. The results are expected to demonstrate that the selected modeling approach substantially influences the accuracy of deadlock analysis in service-oriented systems.

Keywords

Recursion, Concurrency, Recursive Petri Nets, Deadlocks, Service-Oriented Systems

Introduction

As service-oriented architectures are increasingly adopted in modern software systems, ranging from intelligent IoT devices in smart homes to complex robotics systems based on open-source frameworks such as ROS [1], the analysis of such systems is becoming increasingly important. A particular challenge in this context is deadlock analysis: deadlocks often occur in unpredictable ways and their causes are difficult to identify. This problem is further intensified in distributed service-oriented systems, where the source code of individual services is often inaccessible and the services themselves act autonomously.

This extended abstract presents an abstraction-based approach, top-down, to analysing deadlocks in service-oriented systems with properties such as unbounded concurrency and unbounded recursion. Starting from implemented services, we generate an abstract representation using the model of recursive Petri nets. Each service can provide this abstraction without revealing its internal implementation or source code, thereby enabling deadlock analysis to be performed independently of the actual system code. Previous studies [2] have demonstrated that classical models, such as regular Petri nets, can produce false-positive results. Deadlocks may exist within the system yet remain undetected by the analysis. In contrast, the (G, G)-PRS model [3] has been shown to correctly detect such deadlocks. This work aims to investigate whether recursive Petri nets can similarly preserve existing deadlocks and support precise deadlock detection. However, we will, based on the notion of [4], formally introduce recursive Petri nets (RPNs), present the abstraction and provide a formal definition of deadlocks in recursive Petri nets. We will then discuss the resulting implications for deadlock analysis.

Deadlocks in recursive Petri nets and service abstraction

The notation and semantics of recursive Petri nets (RPNs) used in this work follow the definitions and notation established by Haddad and Poitrenaud in [4], including those for extended markings. In analogy to the language of RPNs [4], a marked recursive Petri net is defined with initial and final *extended* markings, whereas in other definitions, final markings and the associated *cut*-step have to be considered separately [5], [6], [7].

PNSE'25: International Workshop on Petri Nets and Software Engineering, June 23–24, 2025, Paris, France

*Corresponding author.

†These authors contributed equally.

erik.hartnick@student.uni-halle.de (E. J. Hartnick); mandy.weissbach@informatik.uni-halle.de (M. Weißbach)

0009-0003-2377-7349 (E. J. Hartnick); 0009-0007-7458-3658 (M. Weißbach)

Definition 1 (Marked Recursive Petri Net)**.** A marked recursive Petri net (with initial extended marking and final extended markings) is the quadruple $\langle N, tr_0, Tr_f, tr\rangle$, where

(i) $N \triangleq \langle P, T, W^-, W^+, \Omega\rangle$ is a recursive Petri net
(ii) $tr_0 \triangleq \langle V_0, M_0, E_0, A_0\rangle$ is an initial extended marking of N
(iii) $Tr_f \triangleq \{tr_f \mid tr_f = \langle V_f, M_f, E_f, A_f\rangle\}$ is a set of final extended markings of N and
(iv) $tr \triangleq \langle V, M, E, A\rangle$ is an extended marking of N reachable from tr_0 by $tr_0 \xrightarrow{\sigma} tr$, $\sigma = t_1 t_2 \cdots t_n$.

The firing of a transition $t \in T$, $tr \xrightarrow{t} tr'$ results in a marked recursive Petri net $\langle N, tr_0, Tr_f, tr'\rangle$, provided that $tr \notin Tr_f$. If $tr \in Tr_f$, execution halts regardless of any enabled transistions. Note that an initial marking refers to $\Omega(t_{ab})$ for $t_{ab} \in T_{ab}$, whereas an initial *extended* marking refers to tr_0.

For each vertex $v \in V$ in an extended marking tr of N, the associated marking $M(v)$ is referred to as a thread [4]. A deadlock that is local to a thread $M(v)$ is therefore termed a threadlock.

Definition 2 (Threadlock)**.** Let $\langle N, tr_0, Tr_f, tr\rangle$ be a marked recursive Petri net with the current extended marking $tr \triangleq \langle V, M, E, A\rangle$ and $tr \notin Tr_f$. A thread $M(v)$ of vertex $v \in V$ in the extended marking tr is in a *threadlock*, iff no transitions $t \in T$ are enabled in $M(v)$, i.e. $\forall t \in T : \exists p \in P : M(v)(p) < W^-(p, t)$.

A threadlock also constitutes a deadlock in the underlying Petri net $\bar{N} \triangleq \langle P, T, W^-, W^+\rangle$, marked by $M(v)$. A deadlock in a recursive Petri net can thus be constructed from individual threadlocks.

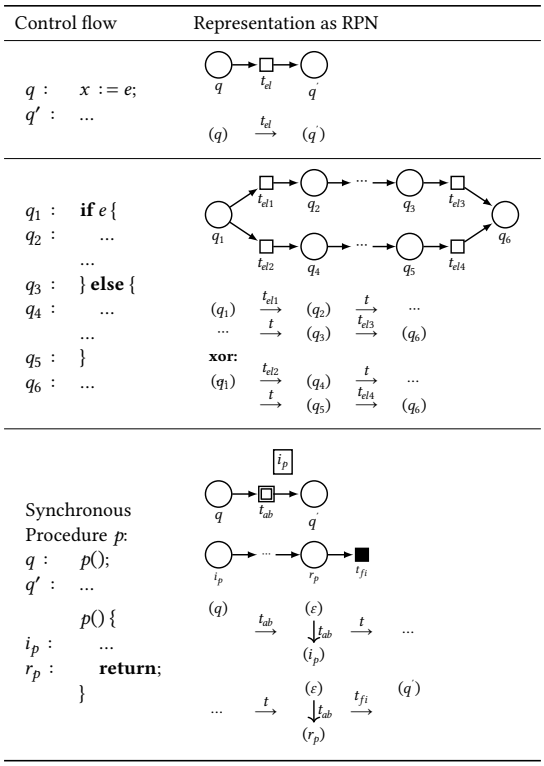
Definition 3 (Deadlock in a RPN)**.** Let $\langle N, tr_0, Tr_f, tr\rangle$ be a marked recursive Petri net, let $tr \triangleq \langle V, M, E, A\rangle$. The marked recursive Petri net is in a *deadlock* for tr iff $tr \notin Tr_f$ and for all $v \in V$, $M(v)$ is in a threadlock.

Based on these two types of deadlocks, the abstraction from the service-oriented language $\mathcal{XYZ}$ is described, as illustrated in Tables Table 1a and Table 1b. For each control flow element shown in the left column, the corresponding recursive Petri net fragment is depicted in the upper part of the right column, while a sequence of extended markings is presented in the lower part. To simplify the notation, threads $M(v)$ are expressed using process-algebraic expressions of class P according to Mayr's hierarchy [3], where place names are separated by the commutative and associative operator $\|$.

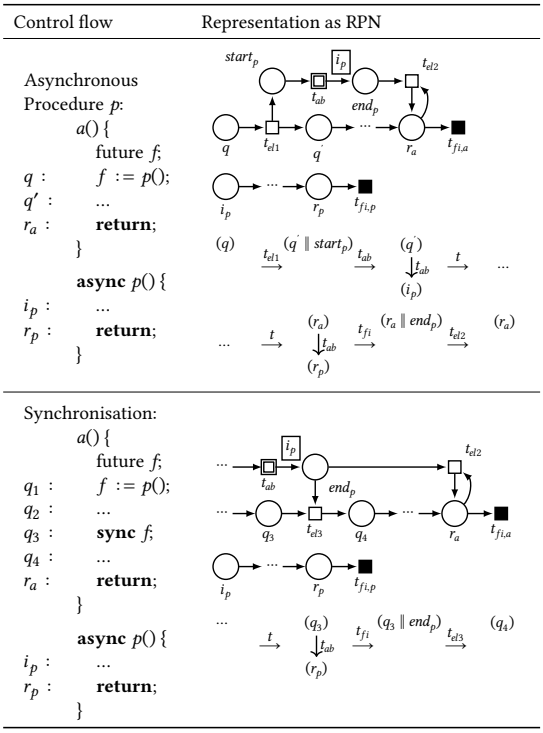
The number of occurrences of a place in a process-algebraic expression corresponds to the number of tokens in that place, representing asynchronous execution in a $(P, P) - PRS$. The places $start_p$ and end_p are introduced to ensure that not all tokens from previous places are held until a final transition in the child thread fires. As a result, an asynchronous procedure call cannot be modeled using an abstract transition alone. Since recursive Petri nets are bipartite graphs, direct edges between transitions are not permitted; places $start_p$ and end_p must be inserted accordingly.

Conclusions

This extended abstract presented an abstraction-based approach to enable deadlock analysis in service-oriented architectures. The approach builds on an established programming model introduced in prior work [2]. A formal definition of deadlocks in recursive Petri nets (RPNs) was developed within the context of this abstraction. Existing counterexamples must be examined more thoroughly to support the validation of the proposed method. This will help assess whether the approach extends the expressive power of existing techniques, even in cases where certain deadlock scenarios—such as threadlocks—may remain undetected. Future work includes the identification and analysis of additional deadlock types in RPNs. Moreover, the ability of RPNs to model error handling is to be investigated. Prior studies have shown that correct modeling of exception handling is only achievable through (G,G)-PRSs [8]. It remains to be examined whether RPNs exhibit similar capabilities, potentially positioning them between PANs and (G,G)-PRSs within the Mayr hierarchy [3]. In addition, the backward inclusion relation between PANs and RPNs will be formally analysed to determine whether PANs form a proper subset of RPNs. Parallel to these theoretical investigations, tooling support for deadlock analysis in RPNs is under development, aiming to bridge the gap between theoretical insights and practical application.



(1a) Abstraction of assignment, if-else and synchronous procedure call control flows to recursive Petri nets.



(1b) Abstraction of asynchronous procedure call control flows to recursive Petri nets.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, et al., Ros: an open-source robot operating system, in: ICRA workshop on open source software, volume 3, Kobe, 2009, p. 5.

[2] M. Weißbach, W. Zimmermann, On Limitations of Abstraction-Based Deadlock-Analysis of Service-Oriented Systems, Springer International Publishing, 2020, pp. 79–90. doi:10.1007/978-3-030-63161-1_6.

[3] R. Mayr, Process rewrite systems, Information and Computation 156 (2000) 264–286. URL: https://www.sciencedirect.com/science/article/pii/S0890540199928262. doi:10.1006/inco.1999.2826.

[4] S. Haddad, D. Poitrenaud, Theoretical Aspects of Recursive Petri Nets, Springer Berlin Heidelberg, 1999, pp. 228–247. doi:10.1007/3-540-48745-x_14.

[5] S. Haddad, D. Poitrenaud, Modelling and Analyzing Systems with Recursive Petri Nets, Springer US, 2000, pp. 449–458. doi:10.1007/978-1-4615-4493-7_48.

[6] S. Haddad, D. Poitrenaud, Recursive petri nets, Acta Informatica 44 (2007) 463–508. doi:https://doi.org/10.1007/s00236-007-0055-y.

[7] A. Finkel, S. Haddad, I. Khmelnitsky, Coverability and Termination in Recursive Petri Nets, Springer International Publishing, 2019, pp. 429–448. doi:10.1007/978-3-030-21571-2_23.

[8] C. Heike, W. Zimmermann, A. Both, On expanding protocol conformance checking to exception handling, Service Oriented Computing and Applications 8 (2014) 299–322. doi:10.1007/s11761-013-0146-2.