On Abstraction-Based Deadlock-Analysis in Service-Oriented Systems with Recursion

Mandy Weissbach, Wolf Zimmermann

On Abstraction-Based Deadlock-Analysis in Service-Oriented Systems with Recursion

Mandy Weißbach and Wolf Zimmermann

Martin Luther University Halle-Wittenberg, Institute of Computer Science,
Von-Seckendorff-Platz 1, 06120 Halle, Germany
{mandy.weissbach,wolf.zimmermann}@informatik.uni-halle.de

Abstract. We examine deadlock analysis for service-oriented systems with unbound concurrency and unbound recursion. In particular, abstraction-based approaches are considered, i.e., abstract behavior models are derived from service implementations and composed according to the architecture of service-oriented systems. It turns out that there are some limitations of Petri-net-based approaches, e.g., such as workflow nets if deadlocks are analyzed. We show an example that ends in a deadlock if recursion is considered but on a Petri-net-based abstraction, it may regularly end.

Keywords: Process Rewrite Systems; Deadlock; Workflow Nets.

1 Introduction

To reduce the risk of unintended behavior (e.g., deadlocks or livelocks [14]) of service-oriented systems due to composition, many approaches are proposed, e.g., protocol conformance checking [2], [10], [11] or deadlock analysis [13].

In this paper we focus on an abstraction-based approach for deadlock analysis of service-oriented systems including concurrency and recursion.

Approaches, e.g., van der Aalst's workflow nets [13] are Petri-net-based and used to analyze deadlocks. They do not consider recursion, recursive callbacks and synchronization. These approaches are refinement-based, i.e., the behavior of a service is modeled as a workflow net and then refined to the service implementation. Workflow nets are used to check for the absence of deadlocks. In contrast, we provide an abstraction-based approach, i.e., the behavior is automatically abstracted from the service's implementation using classical compiler technologies [1] covering all kinds of programming concepts (synchronous and asynchronous procedure calls, synchronization, cf. Tab. 1). Motivation for an abstraction-based approach is that there are many services not developed according to a refinement-based approach. Furthermore, even if they have been developed initially by a refinement-based approach, it is unlikely that programmers consistently maintain the implementation and its abstraction.

In [15] it was shown that abstraction from recursion may lead to false positives for protocol conformance checking. In this work, we examine the same question for deadlock analysis. We compare Petri-net-based abstractions with

abstractions including recursion. The behavior of recursive procedures and synchronous procedure calls corresponds to the LIFO principle and requires therefore a stack [8] to trace the calling context. Process rewrite systems (PRSs) are an extension of Petri nets by stacks [9] and therefore PRS allow to model the behavior of (recursive) procedure calls, concurrency (fork), synchronization and exception handling [6].

Furthermore, [6] shows that there is a correspondence between process algebraic expressions defined by an abstraction based on process-algebras and cactus stacks (introduced as tree of stacks by [4]). Therefore, we focus on PRSs which include pushdown systems as well as Petri nets. Checking reachability and deadlocks remains decidable in process rewrite systems [9].
Our main results are:

- Each trace of a process rewrite system based abstraction corresponds step by step to a trace of the corresponding Petri-net-based abstraction.
- A (reachable) deadlock in the process rewrite system based abstraction does not necessarily correspond to a deadlock in the corresponding Petri-net-based abstraction.

This paper is organized as follows: In Section 2 we introduce service-oriented systems, Mayr's process rewrite systems according to [9] and we show the abstraction and composition process of a service-oriented system including unbound concurrency and unbound recursion. Section 3 discusses the correspondence between Petri net and process rewrite system abstractions. Furthermore, it shows that reachable deadlocks in the process rewrite system based abstraction do not correspond to deadlocks in the corresponding Petri-net-based abstraction. Section 4 discusses the related work and Section 5 concludes with a short overview of the results and gives an outlook.

2 Foundations

2.1 Services and Service-Oriented Systems

A service-oriented system is composed by two or more services which communicate over a required and provided interface, cf. Fig. 1. We assume that a service A is an implementation with a provided interfaces I_A, where an interface is a set of procedure signatures. The required interface R_s of service S is the set of procedures of other services called by S, cf. Fig. 1. It is possible that a service calls a procedure of other services, e.g., service S calls the required procedure a of service A provided by the provided interface I_A.

Procedures of an interface can be either called synchronously (procedure a of interface I_A) or asynchronously (procedure b of interface I_B). If a synchronous procedure is called, it blocks the caller until the callee has been completed. If an asynchronous procedure is called then the callee and the caller continue their execution in parallel. They are either synchronized by an explicit statement (**sync**, program point q_{a6} of service A) on the caller site or when both, caller and callee reach their return statement, cf. Fig. 1 r_a of service A.

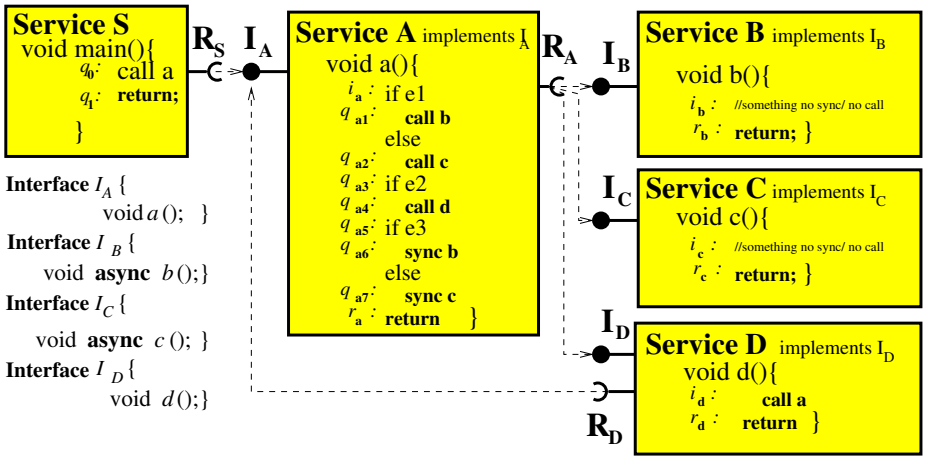

Fig. 1: A Service-Oriented System with Services S, A, B, C and D. Service S acts as a client. Procedure b, c are asynchronous and a, d synchronous procedures.

$$\frac{e \to e'}{e \Rightarrow e'} \text{ (R)} \qquad \frac{e \Rightarrow e'}{e.s \Rightarrow e'.s} \text{ (S)} \qquad \frac{e \Rightarrow e' \quad e' \Rightarrow e''}{e \Rightarrow e''} \text{ (T)}$$

$$\frac{e \Rightarrow e'}{e \parallel s \Rightarrow e' \parallel s} \text{ (P1)} \qquad \frac{e \Rightarrow e'}{e \parallel s \Rightarrow e' \parallel s} \text{ (P2)} \qquad \frac{}{u \Rightarrow u} \text{ (L)}$$

$e, e', e'', s \in PEX(Q)$

Fig. 2: Inference Rules for the Definition of the Derivation Relation in a PRS

2.2 Process Rewrite Systems

Mayr presented a unified view of Petri nets and several simple process algebras by representing them as subclasses of the general rewriting formalism *Process Rewrite Systems* [9]. It is based on rewrite rules on process-algebraic expressions. The set $PEX(Q)$ of process-algebraic expressions over a finite set Q (*atomic processes*) is the smallest set satisfying:

(i) $Q \subseteq PEX(Q)$,

(ii) If $e, e' \in PEX(Q)$, then $e.e' \in PEX(Q)$ and $e \parallel e' \in PEX(Q)$ (*sequential* and *parallel composition*, respectively).

The parallel composition is associative and commutative. The sequential composition is associative but not commutative.

Definition 1 (Process Rewrite Systems) *A* process rewrite system *(short: PRS) is a tuple* $\Pi \triangleq (Q, q_0, \to, F)$ *where*

(i) Q *is a finite set (*atomic processes*),*

(ii) $q_0 \in Q$ *(the* initial state*, an atomic process),*

(iii) $\to \subseteq PEX(Q) \times PEX(Q)$ *is a set of* process-rewrite rules*,*

(iv) $F \subseteq Q$ *(the set of* final processes*).*

The PRS Π *defines a derivation relation* $\Rightarrow \subseteq PEX(Q) \times PEX(Q)$ *by the inference rules in Fig. 2.*

Control Structure	Abstraction	Control Structure	Abstraction
q_i : *assignment*; q_j : $\cdots$	**(G,G)** $q_i \to q_j$ **(P,P)** $q_i \to q_j$	Synchronization q_i : **sync** b; q_{i+1} : $\cdots$ $b\{$ $\cdots$ q_j : **return**$\}$	**(G,G)** $q_i \parallel q_j \to q_{i+1}$ **(P,P)** $q_i \parallel q_j \to q_{i+1}$
q_i : **while** $e\{$ q_j : $\cdots\}$ q_k : $\cdots$	**(G,G)** $q_i \to q_j$ $q_i \to q_k$ **(P,P)** $q_i \to q_j$ $q_i \to q_k$	Synchronous procedure a q_i : **call** a; q_{i+1} : $\cdots$ $a\{q_j$: $\cdots$ q_k : **return**$\}$	**(G,G)** $q_i \to q_j \cdot q_{i+1}$ $q_k \cdot q_{i+1} \to q_{i+1}$ **(P,P)** $q_i \to q_j$ $q_k \to q_{i+1}$
q_i **if** $e\{$ q_j $\cdots$ q_k *last program point*$\}$ **else**$\{$ q_l $\cdots$ q_m *last program point*$\}$ q_n $\cdots$	**(G,G)** $q_i \to q_j$ $q_i \to q_l$ $q_k \to q_n$ $q_m \to q_n$ **(P,P)** $q_i \to q_j$ $q_i \to q_l$ $q_k \to q_n$ $q_m \to q_n$	Asynchronous procedure b $a\{$ $\cdots$ q_i **call** b; q_{i+1} $\cdots$ q_j **return** $\}$ $\cdots$ $b\{$ q_k : $\cdots$ q_l : **return**$\}$	**(G,G)** $q_i \to q_{i+1} \parallel q_k$ $q_j \parallel q_l \to q_j$ **(P,P)** $q_i \to q_{i+1} \parallel q_k$ $q_j \parallel q_l \to q_j$

Table 1: Control-Flow Abstractions to (G,G)-PRS and (P,P)-PRS

PRSs where no rule contains a sequential composition operator ((P,P)-PRS) are equivalent to Petri nets [9]. Hence, the following definition applies to general process rewrite systems ((G,G)-PRS) as well as to Petri nets.

Definition 2 *Let $\Pi = (Q, q_0, \to, F)$ be a PRS. A process algebraic expression $e \in PEX(Q)$ is* reachable *iff $q_0 \Rightarrow e$. A reachable $e \in PEX(Q)$ is a* deadlock *iff there exists no $e' \in PEX(Q) \setminus F$, $e' \neq e$ such that $e \Rightarrow e'$.*

2.3 Abstraction and Composition Process

Table 1 shows different control structures and their abstraction to (P,P)-PRS and (G,G)-PRS. The main principle is that each statement corresponds to a program point (which refers to a statement). The most important control structures are contained in Table 1, atomic statements, e.g., assignments, conditionals, synchronous and asynchronous procedure calls and synchronizations. Loops and case statements are abstracted similarly to conditionals. For service-oriented abstractions, the control-flow abstraction rules can be applied to every services. The main difference is that entry and exit points are n eeded for the first program point and the return statement of the procedure of a required interface of a service. These entry and exit points are identified upon composition with the corresponding services implementing the required interface. This combination yields to a PRS modeling an abstract behavior of the service-oriented system, cf. [2]. An analogous idea is used in [13] for combining workflow nets to Petri nets representing the behavior of the composed service-oriented system.

Example 1 (A Service-Oriented System and its Abstractions). The example in Fig. 1 was introduced in Subsection 2.1. Figure 3 shows the abstraction of the single services using the entry points i_a, i_b, i_c, i_d and the exit points r_a, r_b, r_c, r_d for the initial program points and the program points of the return statements of a, b, c, d, respectively. The final state of the PRS is q_1. Figure 3 shows the resulting abstractions for (G,G)-PRS and (P,P)-PRS, respectively.

Source Code of Fig. 1	(G,G)-PRS	(P,P)-PRS
$main\{$ q_0 : **call** a q_1 : **return**; $\}$	$q_0 \to i_a.q_1$	$q_0 \to i_a$
$a\{i_a$: **if** e_1 q_{a1} : **call** b **else** q_{a2} : **call** c q_{a3} : **if** e_2 q_{a4} : **call** d q_{a5} : **if** e_3 q_{a6} : **sync** b **else** q_{a7} : **sync** c r_a : **return**$\}$	$i_a \to q_{a1}$, $i_a \to q_{a2}$ $q_{a1} \to q_{a3} \parallel i_b$ $r_a \parallel r_b \to r_a$ $q_{a2} \to q_{a3} \parallel i_c$ $r_a \parallel r_c \to r_a$ $q_{a3} \to q_{a4}$, $q_{a3} \to q_{a5}$ $q_{a4} \to i_d.q_{a3}$ $r_d.q_{a5} \to q_{a7}$ $q_{a5} \to q_{a6}$, $q_{a5} \to q_{a4}$ $q_{a6} \parallel r_b \to r_a$ $q_{a7} \parallel r_c \to r_a$ $r_a.q_1 \to q_1$, $r_a.r_d \to r_d$	$i_a \to q_{a1}$ $i_a \to q_{a2}$ $q_{a1} \to q_{a3} \parallel i_b$ $r_a \parallel r_b \to r_a$ $q_{a2} \to q_{a3} \parallel i_c$ $r_a \parallel r_c \to r_a$ $q_{a3} \to q_{a4}$, $q_{a3} \to q_{a5}$ $q_{a4} \to i_d$ $r_d \to q_{a7}$ $q_{a5} \to q_{a6}$, $q_{a5} \to q_{a4}$ $q_{a6} \parallel r_b \to r_a$ $q_{a7} \parallel r_c \to r_a$ $r_a \to q_1$, $r_a \to r_d$
$b\{$ i_b : $calc$(no call/sync) r_b : **return**$\}$	$i_b \to r_b$	$i_b \to r_b$
$c\{$ i_c : $calc$(no call/sync) r_c : **return**$\}$	$i_c \to r_c$	$i_c \to r_c$
$d\{$ i_d : **call** a r_d : **return**$\}$	$i_d \to i_a.r_d$	$i_d \to r_d$

Fig. 3: Abstractions of the Service-Oriented System in Fig. 1

3 Correspondence between (G,G)-PRS and (P,P)-PRS Abstractions

A *run* of process rewrite system $\Pi = (Q, q_0, \to, F)$ is a sequence $e_0, \ldots, e_n$ of process-algebraic expressions such that $e_i \Rightarrow e_{i+1}$, $i = 0, \ldots, n-1$ where $e_i \Rightarrow e_{i+1}$ can be proven without using rules (T) and (L). Intuitively, this means that exactly one PRS-rule is being applied in $e_i \Rightarrow e_{i+1}$ and the sequence $e_0, \ldots, e_n$ represents a step-wise execution of Π. Let S be aservice-oriented system, $\Pi_S \triangleq (Q, q_0, \to_\Pi, F)$ be the (G,G)-PRS abstraction of S and $\Pi'_S \triangleq (Q, q_0, \to_{\Pi'}, F)$ the (P,P)-PRS abstraction of S, cf. Table 1. Note that the set of atomic processes and the initial state is by construction the same in both (G,G)- and (P,P)-PRS. We show that each run of Π_S corresponds to a run in $\Pi_{S'}$.

For this, we need to define an abstraction function α for process-algebraic expressions of Π_S and Π'_S. Since the PRS rules $\to_{\Pi'}$ do not contain the sequential operator the same holds for all reachable expressions. Therefore, the abstraction function $\alpha : PEX(Q) \to PEX(Q)$ forgets the sequential composition, i.e., α is inductively defined by

(i) $\alpha(q) \triangleq q$ for $q \in Q \cup \{\varepsilon\}$
(ii) $\alpha(e_1 \parallel e_2) \triangleq \alpha(e_1) \parallel \alpha(e_2)$ for $e_1, e_2 \in PEX(Q)$
(iii) $\alpha(e_1.e_2) \triangleq \alpha(e_1)$ for $e_1, e_2 \in PEX(Q)$

Example 2 (Runs and Abstractions). The first two columns of Fig. 4 shows a run of the (G,G)-PRS abstraction $\Pi_S = (Q, q_0, \to_\Pi, F)$ and a corresponding run of the (P,P)-PRS abstraction $\Pi'_S = (Q, q_0, \to_{\Pi'}, F)$ of the service-oriented system S in Example 1 (cf. Fig. 1 and Fig. 3). The process algebraic expressions in each row corresponds, i.e., $e'_i = \alpha(e_i)$ where e_i is the first expression (contained in the run in Π_S) of the i-th row and e'_i is second expression (contained in the run in Π'_S) of the i-th row. Furthermore, it holds $\to_{\Pi'} = \{\alpha(e_1) \to_{\Pi'} \alpha(e_2) : e_1 \to_\Pi e_2\}$

(G,G)-PRS	(P,P)-PRS	applied rules (cf. Fig. 3) (G,G)-PRS	(P ,P)-PRS
q_0	q_0		
$i_a.q_1$	i_a	$q_0 \to i_a.q_1$	$q_0 \to i_a$
$q_{a1}.q_1$	q_{a1}	$i_a \to q_{a1}$	$i_a \to q_{a1}$
$(q_{a3} \parallel i_b).q_1$	$q_{a3} \parallel i_b$	$q_{a1} \to q_{a3} \parallel i_b$	$q_{a1} \to q_{a3} \parallel i_b$
$(q_{a4} \parallel i_b).q_1$	$q_{a4} \parallel i_b$	$q_{a3} \to q_{a4}$	$q_{a3} \to q_{a4}$
$(q_{a4} \parallel r_b).q_1$	$q_{a4} \parallel r_b$	$i_b \to r_b$	$i_b \to r_b$
$((i_d.q_{a5}) \parallel q_{12}).q_1$	$i_d \parallel r_b$	$q_{a4} \to i_d.q_{a5}$	$q_{a4} \to i_d$
$((i_a.r_d.q_{a5}) \parallel r_b).q_1$	$i_a \parallel r_b$	$i_d \to i_a.q_{16}$	$i_d \to i_a$
$((q_{a2}.r_d.q_{a5}) \parallel r_b).q_1$	$q_{a2} \parallel r_b$	$i_a \to q_{a2}$	$i_a \to q_{a2}$
$(((q_{a3} \parallel i_c).r_d.q_{a5}) \parallel r_b).q_1$	$q_{a3} \parallel i_c \parallel r_b$	$q_{a2} \to q_{a3} \parallel i_c$	$q_{a2} \to q_{a3} \parallel i_c$
$(((q_{a3} \parallel r_c).r_d.q_{a5}) \parallel r_b).q_1$	$q_{a3} \parallel r_c \parallel r_b$	$i_c \to r_c$	$i_c \to r_c$
$(((q_{a5} \parallel r_c).r_d.q_{a5}) \parallel r_b).q_1$	$q_{a5} \parallel r_c \parallel r_b$	$q_{a3} \to q_{a5}$	$q_{a3} \to q_{a5}$
$(((q_{a6} \parallel r_c).r_d.q_{a5}) \parallel r_b).q_1$	$q_{a6} \parallel r_c \parallel r_b$	$q_{a5} \to q_{a6}$	$q_{a5} \to q_{a6}$

Fig. 4: Runs in the (G,G)-PRS and (P,P)-PRS Abstractions of Fig. 3

Remark 1. A look at Table 1 shows that in general, $\to_{\Pi'} = \{\alpha(e_1) \to_{\Pi'} \alpha(e e_1 \to_{\Pi} e_2\}$, i.e., the rewrite rules of the (P,P)-PRS can be obtained from the rewrite rules of the (G,G)-PRS by forgetting about the sequential composition.

Theorem 1 (Correspondence between Abstractions to (G,G)-PRS and (P,P)-PRS). *Let S be a service-oriented system, $\Pi_S = (Q, q_0, \to_{\Pi}, F)$ be the abstraction of S to (G,G)-PRS according to Table 1, and $\Pi'_S = (Q, q_0, \to_{\Pi'}, F)$ be the abstraction of S to (P,P)-PRS according to Table 1. If $e \Rightarrow_{\Pi} e'$ then $\alpha(e) \Rightarrow \alpha(e')$.*

Proof. The proof is by induction on the number of applications of the inference rules. Suppose $e \Rightarrow_{\Pi} e'$.

Case 1: Rule (R) is being applied. Then $e \to_{\Pi} e'$ according to Remark 1 it is $\alpha(e) \to_{\Pi'} \alpha(e')$.

Case 2: Rule (S) has been applied. Then, $e = e''.s$ and $e' = \bar{e}.s$ for some $e'', \bar{e}, s \in PEX(Q)$, and $e'' \Rightarrow_{\Pi} \bar{e}$. By induction hypothesis, it holds $\alpha(e'') \Rightarrow_{\Pi'} \alpha(\bar{e})$. Now, rule (S) can be applied to obtain $\alpha(e'').s \Rightarrow_{\Pi'} \alpha(\bar{e}).s$. Thus $\alpha(e) \Rightarrow_{\Pi'} \alpha(e')$ using property (iii) of the definition of α.

The cases where rules (P1), (P2), and (T) are applied are proven analogously to Case 2.

Corollary 1. *For each run $e_0, \ldots, e_n$ of Π_S, the sequence $\alpha(e_0), \ldots, \alpha(e_n)$ is a run of Π'_S.*

Hence, each run in the PRS-abstraction corresponds to a run in the (P,P)-PRS abstraction (which is equivalent to the Petri nets). Thus, the workflow nets [13] lead to a coarser abstraction than using general PRS [6].

Now, we examine the deadlock situations. Expression $e \triangleq (((q_{a6} \parallel r_c).r_d.q_{a5}) \parallel r_b).q_1$ is a deadlock because no PRS rule is applicable, cf. Fig. 4. However, the corresponding (P,P)-PRS expression $\alpha(e) = q_{a6} \parallel r_c \parallel r_b$ is not a deadlock. Since $\parallel$ is associative and commutative, it holds

$$q_{a6} \parallel r_c \parallel r_b \overset{\text{ass. and com. } \parallel}{\Longrightarrow} q_{a6} \parallel r_b \parallel r_c \overset{q_{a6} \parallel r_b \to r_a}{\Longrightarrow} r_a \parallel r_c \overset{r_a \to r_d}{\Longrightarrow} r_d \parallel r_c \overset{r_d \to q_{a5}}{\Longrightarrow}$$
$$q_{a5} \parallel r_c \overset{q_{a5} \to q_{a7}}{\Longrightarrow} q_{a7} \parallel r_c \overset{q_{a7} \parallel r_c \to r_a}{\Longrightarrow} r_a \overset{r_a \to q_1}{\Longrightarrow} q_1$$

Thus, the final state q_1 is reached. However, there are alternatives leading to a deadlock. For example the rules $r_a \to r_d$ and $r_d \to q_{a5}$ could be applied to the derivation r_a. This can lead to the deadlock q_{a7}.

4 Related Work

Van der Aalst [13] uses Petri-net-based analysis tool to verify business process workflows. Recursion, e.g., recursive callbacks, is not considered.

In [12] recursive Petri nets (rPNs) are used to model the planning of autonomous agents which transport goods form location A to B. The model of rPNs is used to model dynamic processes (e.g., agent's request). Recursion in our sense is not considered. Deadlocks can only arise when interactions between agents (e.g., shared attributes) invalidates preconditions. Another refinement based approach is described in [7]. Hicheur models healthcare processes based on algebraic and recursive Petri nets [5]. Recursive Petri nets are used to model by the main process called subprocesses. All these approaches use the ability of rPNs to prune subtrees.

Bouajjani et al. [3] work is the closest to ours. They discuss the abstraction-based analysis of recursive parallel programs based on recursive vector addition systems. They explore decidability of reachability for recursively parallel programs. It seems that their model is slightly more general as there are situations where the reachability problem becomes undecidable.

To our knowledge, abstraction-based deadlock analysis in service-oriented systems including synchronous and asynchronous procedure calls (forking), recursion and recursive callbacks and synchronization in the context of service-oriented systems was not investigated before.

5 Conclusion

We examined two different abstractions from service-oriented systems S to general (G,G)-PRS Π_S and to (P,P)-PRS Π' (which are equivalent to Petri nets). We have shown that Π' is more abstract than Π (Theorem 1). However, there is a reachable deadlock e in Π_S where the corresponding situation e' in Π'_S is not necessarily a deadlock although each run $q_0 \to_{\Pi_S} e_1 \to_{\Pi_S} \cdots \to_{\Pi_S} e_n$ in the PRS Π_S has a corresponding run $q_0 \to_{\Pi'_S} e'_1 \to_{\Pi'_S} \cdots \to_{\Pi'_S} e'_n$. To the best of our knowledge, we are not aware on studies on abstraction-based deadlock analysis of service-oriented systems taking into account unbound recursion and unbound concurrency with synchronization.

The main result shows that the Petri net abstraction is too coarse. Furthermore, the example requires recursion. However, in our example the Petri net abstraction Π'_S the final state as well as a deadlock situation is reachable from e'. Therefore, the example doesn't provide a false positive (i.e., it erroneously classifies the service-oriented system S deadlock-free) in the classical sense. Our hypothesis, is that in the context of the paper, if a deadlock situation e in the PRS abstraction Π_S of a service-oriented system S is reachable, then a deadlock situation e'' is reachable from the corresponding situation e' in the Petri net abstraction Π'_S. It is an open question whether this hypothesis is true. However,

even it is true, the trace leading to a deadlock situation e'' cannot be obtained by execution of S. This may erroneously lead to classify the deadlock e'' as a false alarm.

References

1. Aho, A.V., Lam, M.S., Sethi, R., Ullman, J.D.: Compilers: Principles, Techniques, and Tools (2nd Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2006)
2. Both, A., Zimmermann, W.: Automatic protocol conformance checking of recursive and parallel component-based systems. In: Component-Based Software Engineering, 11th International Symposium (CBSE 2008). pp. 163–179 (October 2008)
3. Bouajjani, A., Emmi, M.: Analysis of recursively parallel programs. In: ACM SIGPLAN Notices. vol. 47, pp. 203–214. ACM (2012)
4. Dahl, O.J., Nygaard, K.: Simula: an algol-based simulation language. Communications of the ACM 9, 671–678 (1966)
5. Haddad, S., Poitrenaud, D.: Modelling and analyzing systems with recursive petri nets. In: Discrete Event Systems, pp. 449–458. Springer (2000)
6. Heike, C., Zimmermann, W., Both, A.: On expanding protocol conformance checking to exception handling. Service Oriented Computing and Applications 8(4), 299–322 (2014)
7. Hicheur, A., Dhieb, A.B., Barkaoui, K.: Modelling and analysis of flexible healthcare processes based on algebraic and recursive petri nets. In: Foundations of Health Information Engineering and Systems, pp. 1–18. Springer (2012)
8. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to automata theory, languages, and computation, 2nd edition. SIGACT News 32(1), 60–65 (Mar 2001), http://doi.acm.org/10.1145/568438.568455
9. Mayr, R.: Process rewrite systems. Information and Computation 156(1-2), 264–286 (2000)
10. Parizek, P., Plasil, F.: Objects, Components, Models and Patterns: 46th International Conference, TOOLS EUROPE 2008, Zurich, Switzerland, June 30 - July 4, 2008. Proceedings, chap. Modeling of Component Environment in Presence of Callbacks and Autonomous Activities, pp. 2–21. Springer Berlin Heidelberg, Berlin, Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-69824-1_2
11. Schmidt, H.W., Krämer, B.J., Poernomo, I., Reussner, R.: Predictable component architectures using dependent finite state machines. In: Radical Innovations of Software and Systems Engineering in the Future, pp. 310–324. Springer (2004)
12. Seghrouchni, A.E.F., Haddad, S.: A recursive model for distributed planning. In: Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS'96). pp. 307–314 (1996)
13. Van Der Aalst, W.M.: Workflow verification: Finding control-flow errors using petri-net-based techniques. In: Business Process Management, pp. 161–183. Springer (2000)
14. Weißbach, M., Zimmermann, W.: Termination analysis of business process workflows. In: Proceedings of the 5th International Workshop on Enhanced Web Service Technologies. pp. 18–25. WEWST '10, ACM, New York, NY, USA (2010), http://doi.acm.org/10.1145/1883133.1883137
15. Zimmermann, W., Schaarschmidt, M.: Automatic checking of component protocols in component-based systems. In: Löwe, W., Südholt, M. (eds.) Software Composition. LNCS, vol. 4089, pp. 1–17. Springer (2006)