Verifying Deadlock- and Livelock Freedom in an SOA Scenario

Karsten Wolf*, Christian Stahl[†‡], Janine Ott[‡] and Robert Danitz[‡]
*Institut für Informatik
Universität Rostock, 18051 Rostock, Germany
Email: karsten.wolf@uni-rostock.de
[†]Department of Mathematics and Computer Science
Technische Universiteit Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
[‡] Institut für Informatik
Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany
Email: stahl, jott, danitz@informatik.hu-berlin.de

Abstract

In a service-oriented architecture (SOA), a service broker assigns a previously published service (stored in a service registry) to a service requester. It is desirable for the composition of the requesting and the assigned service to interact properly. While proper interaction is often reduced to deadlock freedom of the composed system, we additionally consider livelock freedom as a desirable property for the interaction of services. In principle, deadlock- and livelock freedom can be verified by inspecting the state space of the composition of (public views of) the involved services.

The contribution of this paper is to propose a methodology to build that state space from pre-computed fragments *which are computed upon publishing a service. That way, we shift computation time from the time critical "request" phase of service brokerage to the less critical "publish" phase. Interestingly, our setting enables state space reduction methods that are intrinsically different from traditional state space reductions.*

1. Introduction

1.1. Background and motivation

A service-oriented architecture (SOA) defines three roles for service owners. A service *provider* offers the functionality of his service by publishing it (or an abstract version, the *public view*) in a service registry. A service *broker* manages the registry. A service *requester* queries the broker for finding a certain functionality. If a fitting service can be found in the registry, the broker returns relevant data for establishing a connection between provided and requesting service. As each service is published only once, but potentially considered for matching multiple times, we may expect that the number of "find" requests at a broker is significantly greater than the number of "publish" events. Moreover, answering a "find" *request* appears to be more time critical than processing a *publish* event since the "find" request may be much closer in time to the actual execution of the composed system.

Selecting a fitting service (*matchmaking*) is a complex task. Apart from the general functionality, the broker must take care of semantical, behavioral, and non-functional aspects. In this article, we contribute to the behavioral aspects of matchmaking which include proper interaction between the matched services. In most articles on this issue [1], [2], deadlock freedom is used as a definition of proper interaction. However, livelock freedom (a livelock is a set of states that cannot be escaped and does not contain a terminal state) is undisputably a desirable property as well. While relevant service description languages like WS-BPEL [3] avoid deadlocks and livelocks in single services, they cannot prevent them in service compositions [4].

In a naive approach to the prevention of deadlock- and livelock freedom, the service broker would construct the transition system that reflects the composed behavior of the candidate services to be matched and model checks it for absence of deadlocks and livelocks. In the branching time logic CTL, the property to be checked could be formalized to $AGEF\ terminalState$, i.e., from every path starting in the initial state, we eventually reach a terminal state. However, this procedure would be executed upon a "find" request. We argued above that this "find" request is rather time critical. For this reason, we wish to shift computation time from the execution of a "find" request to the execution of a "publish" event.

1.2. Approach

Let P be a published service and R a requesting service for which P is a candidate to be matched. As services interact asynchronously, the composition $P \oplus R$ interleaves transitions of P and R. In essence, our idea is to pre-compute, already upon publishing P, those parts of $P \oplus R$ that take place between two subsequent transitions of R. We call these parts *fragments*. We shall show, that there is a finite set of fragments from which arbitrary composed

systems involving P can be built. The set is in particular independent of any R to be composed with P.

At "find", we glue fragments to the actual transition system $P \oplus R$. Here, R determines the way in which fragments are glued. The advantage of this approach is that we can, at "publish" time, apply substantial state space reduction to the fragments. This computation time is no longer consumed during the time critical "find" phase.

Interestingly, the described setting calls for a way of state space reduction that differs significantly from standard model checking approaches. In model checking [5], it typically does not make much sense to condense state spaces a posteriori, i.e., after their actual calculation. In our setting, however, it makes a lot of sense to condense the internal state space of fragments once they have been computed. The reason is that their actual integration into a complete transition system takes place at a different point in time. For a posteriori state space reduction, we use a set of reduction rules adapted from [6].

The paper is structured as follows. We provide our service model and other basic concepts in Sect. 2. Section 3 introduces how to compute fragments of a service P and how to construct a transition system $P \oplus R$ given the fragments of P and a service R. Reduction rules to condense fragments are presented in Sect. 4. A case study in Sect. 5 validates the applicability of our methodology. Section 6 presents related work and conclusions are drawn in Sect. 7.

2. Basic concepts

This section describes our service model, finite state machines, simulation between service models and the notion of a strategy.

2.1. Behavioral models for services

We model the behavior of a service P as a finite state machine $P = [Q, L, \delta, q_0, Q_F]$ consisting of a set Q of states, a (possibly infinite) set L of labels, labeled transitions $\delta \subseteq Q \times L \times Q$, an initial state q_0, and a set $Q_F \subseteq Q$ of final states. A transition may represent an internal activity (labeled τ), sending of message x to the environment (labeled $!x$), or the receipt of a message x from the environment (labeled $?x$). Final states represent successful termination. We require that all transitions leaving a final state must be receive transitions (i.e., there is no activity triggered by the service itself, but reactions to external events are permitted). We use indices to distinguish ingredients of different services whenever necessary.

For two services P and R, their composition $P \oplus R$ is a transition system where states are triples $[q_P, q_R, M]$ consisting of a state q_P of P, a state q_R of R, and a multiset (i.e., bag) M of pending messages. Using this structure, we implement asynchronous communication. The initial state $[q_{0P}, q_{0R}, [\,]]$ consists of the two initial states q_{0P} and q_{0R} of P and R, and the empty multiset $[\,]$. A send transition $!x$ adds 1 to the multiplicity of the sent message x in the bag M; a receive event $?x$ is only enabled if the multiplicity of the involved message x is greater than 0 and, upon occurrence, diminishes this multiplicity by 1. An internal transition τ does not effect the bag of pending messages. No transition of one service changes the state of the other service. Final states of the composed system are those where both services are in their respective final states, and the message bag is empty.

We assume that transitions in $P \oplus R$ inherit their label from the originating transition in P and R, respectively. In contrast to actual service models, however, a label in a composed system just serves as an annotation and is not meant to establish communication with a third service. The annotated labels will significantly simplify some considerations below.

Composition of services may lead to an infinite state system, due to an unbounded growth of the message bag. For keeping our approach tractable, we rule out infinite behavior in the following way. We assume that there is a commonly agreed number k such that any interaction that leads to more than k pending messages of a kind (i.e., a multiplicity larger than k in some message bag) is considered to be erroneous. In service models with practical background available to us, we can rarely find examples where more than one pending message of a kind makes any sense.

Examples for service models are P, R and $MP(\mathsf{P})$ depicted in Figs. 1(a), 1(b) and 1(d), respectively. The composition of P and R is shown in Fig. 1(c).

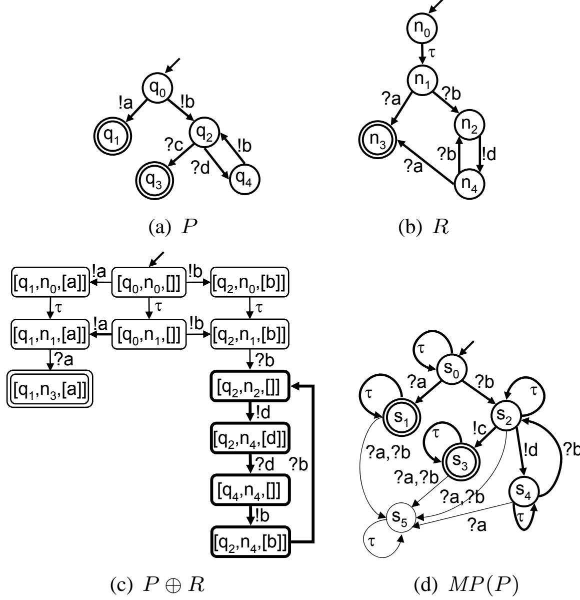
2.2. Simulation between services

For comparing two services R_1 and R_2, we use the well-known concept of a *simulation relation* [7]. A binary relation $\varrho \subseteq Q_{R_1} \times Q_{R_2}$ is a simulation relation of R_1 by R_2 if and only if $[q_{0R_1}, q_{0R_2}] \in \varrho$ and, for all $[q_1, q_2] \in \varrho$, $[q_1, l, q_1'] \in \delta_{R_1}$ implies existence of a q_2' holding $[q_2, l, q_2'] \in \delta_{R_2}$ and $[q_1', q_2'] \in \varrho$. If such a ϱ exists, we say that R_2 *simulates* R_1. If ϱ^{-1} is a simulation relation of R_2 by R_1 as well, we call the two services *bisimular* [8].

Observe that we use a strong kind of simulation where even τ is treated as a normal label. In the example of Fig. 1, service $MP(\mathsf{P})$ simulates service R but obviously, R does not simulate $MP(\mathsf{P})$.

2.3. Strategies

We call R a *strategy* for P if and only if $P \oplus R$ neither contains deadlocks nor livelocks. In other words, every terminal strongly connected component of $P \oplus R$ is required to contain a final state. This property can be verified using CTL model checking with the specification $AG\,EF\,final$ (i.e., from every path starting in the initial



(a) P (b) R

(c) $P \oplus R$ (d) $MP(P)$

Figure 1. (a) Service P and (b) service R, (c) their composition $P \oplus R$, and (d) the most permissive strategy $MP(P)$ for P. Final states are depicted by double circles (boxes).

state, it is always possible to reach a final state). Notice that our definition of a deadlock differs from the standard definition in literature as we discriminate between final states and deadlocks. Consequently, service R is no strategy for service P in Fig. 1, because their composition may livelock (see the four states depicted bold) if P sends message !b.

From [9], we import a result concerning strategies of services.

Proposition 1 (Existence of a most permissive strategy). For every service P which has at least one strategy, there exists a *most permissive strategy* $MP(P)$ such that the following holds: If R is a strategy for P, then $MP(P)$ simulates R. ⌟

There may be several most permissive strategies for a service. The results of this paper hold independently from the most permissive strategy chosen. In [9], we describe an algorithm for computing a most permissive strategy $MP(P)$ for a given P and a given bound k of pending messages in the composition $P \oplus MP(P)$.

Once having found a most permissive strategy, Proposition 1 gives us a necessary but unfortunately not a sufficient criterion for deciding if R is a strategy of P. It does, however, provide us with the clue to our approach: if every strategy R is simulated by a most permissive strategy $MP(P)$, its composition $P \oplus R$ must follow those patterns which are already present in $P \oplus MP(P)$.

The most permissive strategy $MP(\mathsf{P})$ for service P is depicted in Fig. 1(d). Notice that an infinite exchange of messages b and d is possible in the composition of $MP(\mathsf{P})$ with P, because $AG\,EF\,\mathit{final}$ only guarantees that a final state *can* be reached (if $MP(\mathsf{P})$ sends c). State s_5 is depicted for technical purposes only. Every edge to s_5 shows a possible set of messages $MP(\mathsf{P})$ can receive, but that will never occur, because P cannot send them. Although $MP(\mathsf{P})$ simulates R (cf. Fig. 1(b)), R is not a strategy for P.

3. Composing transition systems from fragments

This section defines the notion of a fragment and shows how to compute fragments for a service P. Then it presents how a transition system $P \oplus R$ can be constructed given the fragments of P and a service R.

3.1. Fragments

A transition system $P \oplus R$ consists of interleaving transitions of P and R. That way, we have a canonical decomposition of the transition system. Suppose we remove all transitions of R. This yields a set of unconnected subgraphs of the transition system. Each subgraph consists of nodes and edges where each edge is a transition of P. Each such subgraph is a *fragment* and a transition of R is *connection* and connects two states of different fragments. According to our approach, we want to compose state spaces from fragments.

Definition 3.1 (Fragment).
A (state space) *fragment* $F = (V, E, \Omega)$ consists of

- a set V of *nodes*,
- a set $E \subseteq V \times L \times V$ of (directed) labeled *edges*, and
- a set $\Omega \subseteq V$ of *final nodes*. ⌟

When composing a state space from fragments, it may happen that we need several copies of one and the same fragment. To this end, we introduce fragment *instances*. Let some fixed set N denote the name space of all fragment instances.

Definition 3.2 (Fragment instance).
Let $n \in N$. An *instance* $F(n)$ of a fragment F is built by renaming the constituents as follows: $v \mapsto [v, n]$, $e = [v_1, x, v_2] \mapsto [[v_1, n], x, [v_2, n]]$. ⌟

Fragment instances are fragments again. For gluing fragments, more precisely, for connecting nodes of different fragment (instances), we use the concept of *connections*.

Definition 3.3 (Connection, instance).
A *connection* C between fragments F_1 and F_2 is a subset of $V_{F_1} \times L \times V_{F_2}$. An *instance* $C(m, n)$ of a connection C is defined as $C(m, n) = \{[[v_1, m], x, [v_2, n]] \mid [v_1, x, v_2] \in C\}$, for $m, n \in N$. ⌟

If C is a connection between fragments F_1 and F_2, then $C(m, n)$ is a connection between $F_1(m)$ and $F_2(n)$.

Given a set of fragments and a set of connections we can build a transition system by connecting states of different fragments according to the connections.

Definition 3.4 (Transition system of fragments).
A set $F_1, \ldots, F_n$ of fragments, together with a set $C_1, \ldots, C_m$ of connections defines the *transition system* $TS = (V, E)$ with

- a set $V = \bigcup_{k=1}^{n} V_{F_k}$ of states and
- a set $E = \bigcup_{i=1}^{n} E_{F_i} \cup \bigcup_{j=1}^{m} C_j$ of (directed) labeled transitions. ⌟

3.2. Fragments and connections for a service P

With the sets $\mathcal{F}(P)$ of fragments and $\mathcal{C}(P)$ of connections, we provide those ingredients from which we can produce composed systems involving P. These sets depend on an arbitrarily chosen but fixed most permissive strategy $MP(P)$ for P which, as already mentioned, can be computed from P and a fixed bound k that limits the number of pending messages. Our results do not depend on the particular choice of $MP(P)$.

Definition 3.5 ($\mathcal{F}(P)$, $\mathcal{C}(P)$).
Let $MP(P)$ be the most permissive strategy for P. Define $\mathcal{F}(P) = \{F_q \mid q \in Q_{MP}\}$ and $\mathcal{C}(P) = \{C_{[q,x,q']} \mid [q,x,q'] \in \delta_{MP}\}$, where

- $V_{F_q} = \{[q_P, q, M] \mid [q_P, q, M] \in Q_{P \oplus MP}\}$,
- $E_{F_q} = \delta_{P \oplus MP} \cap [V_{F_q} \times L \times V_{F_q}]$,
- $\Omega_{F_q} = \{[q_P, q, M] \in V_{F_q} \mid q_P \in Q_{FP}, q \in Q_{FMP}, M = [\,]\}$, and
- $C_{[q,x,q']} = \delta_{P \oplus MP} \cap [V_{F_q} \times \{x\} \times V_{F_{q'}}]$. ⌟

For each state $q \in Q_{MP}$, there is a fragment F_q. Informally, fragment F_q describes the behavior of P while $MP(P)$ is in state q. This yields the set of states and edges of F_q. Final states of a fragment are those states, where both P and $MP(P)$ are in a final state and M is the empty multiset. A connection $C_{[q,x,q']}$ describes a transition in the composed system that originates from a single transition $[q, x, q']$ in $MP(P)$.

Due to the introduced bound k of pending messages, both $\mathcal{F}(P)$ and $\mathcal{C}(P)$ are finite.

The fragments and the connections of our example service P are depicted in Fig. 2. For each state s (except the unreachable state s_5), there is one fragment shown in Fig. 2(a). For instance, we have fragment $\mathsf{F}_{s0} = (\{\mathsf{v}_0, \mathsf{v}_1, \mathsf{v}_2\}, \{[\mathsf{v}_0, !\mathsf{a}, \mathsf{v}_1], [\mathsf{v}_0, !\mathsf{b}, \mathsf{v}_2]\}, \emptyset)$ and connection $\mathsf{C}_{[v1,?a,v3]}$. Thereby v_0 relabels $[\mathsf{q}_0, \mathsf{s}_0, [\,]]$, v_1 relabels $[\mathsf{q}_1, \mathsf{s}_0, [\mathsf{a}]]$, etc.

3.3. Composing a transition system from fragments

Throughout this section fix a service P and let $MP(P)$ be the most permissive strategy that has been used for producing $\mathcal{F}(P)$ and $\mathcal{C}(P)$.

By Proposition 1, simulation of a service R by $MP(P)$ is a necessary condition for R being a strategy for P. That is, if $MP(P)$ does not simulate R, R does not interact properly with P (under the given message bound k) and there is no use in constructing a transition system TS_R from R, $\mathcal{F}(P)$ and $\mathcal{C}(P)$ that reflects $P \oplus R$. Thus, we may assume existence of a simulation relation $\varrho \subseteq Q_R \times Q_{MP}$ when constructing TS_R from $\mathcal{F}(P)$ and $\mathcal{C}(P)$. The existence of ϱ is actually checked during the construction of TS_R by relation states of R to fragments of $\mathcal{F}(P)$. The time and space required for finding ϱ is linear in the product of the number of states and edges of $MP(P)$ (i.e., the number of fragments and connections) and R.

For constructing the transition system TS_R, we add for each $[q_R, q] \in \varrho$ a fragment F_q to the set $\mathcal{F}(TS_R)$. More precisely, since q_R might not be the only state of R that is related to q, we add a fragment instance $F_q(q_R)$. Accordingly, the set $\mathcal{C}(TS_R)$ is determined by the connections of P and the fragments of $\mathcal{F}(TS_R)$.

Definition 3.6 (Construction of TS_R).
Let ϱ be a simulation relation of R by $MP(P)$. Compose transition system TS_R from the following fragments and connections:

- $\mathcal{F}(TS_R) = \{F_q(q_R) \mid [q_R, q] \in \varrho\}$,
- $\mathcal{C}(TS_R) = \{C_{[q,x,q']}(q_R, q'_R) \mid [q_R, q] \in \varrho, [q_R, x, q'_R] \in \delta_R, [q, x, q'] \in \delta_{MP}, [q'_R, q'] \in \varrho\}$

Let the set of final states of TS_R consist of all final nodes occurring in those $F_q(q_R)$ where q_R is final, i.e., $q_R \in Q_{FR}$. ⌟

Fragment instance $F_{q_0}(q_{0R})$ is definitely contained in $\mathcal{F}(TS_R)$ and contains an instance of the initial state of $P \oplus MP(P)$. Otherwise, there would be no simulation relation of R by $MP(P)$. We use this particular state as the initial state of TS_R.

In our example, $[\mathsf{n}_0, \mathsf{s}_0]$, $[\mathsf{n}_1, \mathsf{s}_0] \in \varrho$. Thus, we add two instances of F_{s0}, i.e., $\mathsf{F}_{s0}(\mathsf{n}_0)$ and $\mathsf{F}_{s0}(\mathsf{n}_1)$. As we have transition $[\mathsf{n}_0, \tau, \mathsf{n}_1] \in \delta_R$ in R we add connection $\mathsf{C}_{[s0,\tau,s0]}(\mathsf{n}_0,\mathsf{n}_1)$. Furthermore, we add fragment $\mathsf{F}_{s1}(\mathsf{n}_3)$ because $[\mathsf{n}_3, \mathsf{s}_1] \in \varrho$. From transition $[\mathsf{n}_1, ?\mathsf{a}, \mathsf{n}_3] \in \delta_R$ and $[\mathsf{s}_0, ?\mathsf{a}, \mathsf{s}_1] \in \delta_{MP}$ we conclude that connection $\mathsf{C}_{[s0,?a,s1]}(\mathsf{n}_1,\mathsf{n}_3)$ has to be added, and so on. Figure 3 shows the resulting state space TS_R. Observe that $[\mathsf{n}_4, \mathsf{s}_4] \in \varrho$ and transition $?\mathsf{a}$ leaving state n_4 (cf. Fig. 1(b)) yields $[\mathsf{n}_3, \mathsf{s}_5] \in \varrho$. However, as s_5 is not reachable in $MP(\mathsf{P})$, there is no fragment for state s_5, and hence there is no transition $?\mathsf{a}$ leaving $\mathsf{F}_{s4}(\mathsf{n}_4)$. TS_R contains a livelock (the nodes of $\mathsf{F}_{s2}(\mathsf{n}_2)$ and $\mathsf{F}_{s4}(\mathsf{n}_4)$ have no final node). Thus, R is not a strategy for P.

The main result of this section states that the constructed transition system TS_R indeed reflects $P \oplus R$.

Theorem 1.
Let R be a strategy for P which is simulated by $MP(P)$. Let TS_R be as defined above. Then there is a bisimulation

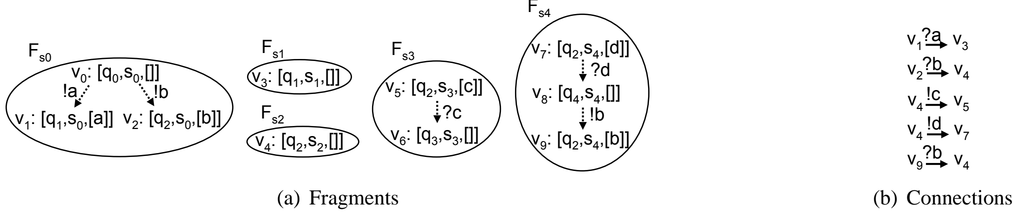


Figure 2. (a) Fragments and (b) connections of P (cf. Fig. 1(a)). Dotted (solid) lines denote fragment internal transitions (connections).

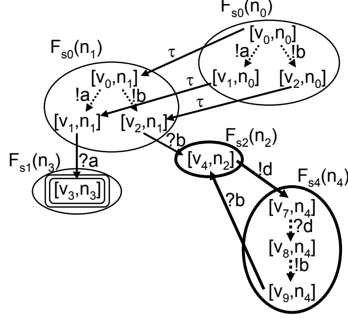


Figure 3. Constructing the state space $P \oplus R$ from the fragments of Fig. 2 for service R of Fig. 1(b). $P \oplus R$ has a livelock (depicted in bold font); the only final state is $[v_3, n_3]$.

between TS_R and $P \oplus R$ that respects final states. ⌟

Proof: Let ϱ be the simulation relation of R by $MP(P)$ that is used for constructing TS_R. We claim that the following relation $\varrho^* \subseteq Q_{P\oplus R} \times Q_{TS_R}$ is the required bisimulation:

$\varrho^* = \{\langle [q_P, q_R, M], [[q_P, q, M], q_R] \rangle \mid [q_P, q_R, M] \in Q_{P\oplus R}, [q_R, q] \in \varrho, [[q_P, q, M], q_R] \in F_q(q_R)\}$

By construction, $F_q(q_R) \in \mathcal{F}(TS_R)$, so ϱ^* is well defined.

The initial state of $P \oplus R$ is $[q_{0P}, q_{0R}, [\,]]$, the initial state of TS_R is $[[q_{0P}, q_0, [\,]], q_{0R}]$, so ϱ^* relates the initial states of the considered systems.

Let $\langle [q_P, q_R, M], [[q_P, q, M], q_R] \rangle \in \varrho^*$. A transition in $P \oplus R$ originates either from a transition in P or a transition in R. A transition of P leads to some state $[q'_P, q_R, M']$ in $P \oplus R$. Obviously, the same transition is possible in state $[q_P, q, M]$ of system $P \oplus MP(P)$ as well, leading to $[q'_P, q, M']$ there. Both states and the corresponding transition between them are thus part of fragment F_q which proves that a transition from $[[q_P, q, M], q_R]$ to $[[q'_P, q, M'], q_R]$ with the same label is present in TS_R. Analogously, a transition internal to a fragment (which again stems from a transition in P) used in TS_R can be copied in $P \oplus R$.

Consider now a transition from state $[q_P, q_R, M]$ to state $[q_P, q'_R, M']$ of $P \oplus R$ that originates from a transition $[q_R, x, q'_R]$ in R. Since ϱ is a simulation, there is a transition $[q, x, q']$ with $[q'_R, q'] \in \varrho$, for some q'. Both transitions change message bag M in the same way since they carry the same label. Fragment $F_{q'}$ and connection instance $C_{[q,x,q']}(q_R, q'_R)$ have been included in the construction of TS_R. This connection instance contains the required transition from $[[q_P, q, M], q_R]$ with x to $[[q_P, q', M'], q'_R]$ in TS_R. The other way round, consider a transition in TS_R from $[[q_P, q, M], q_R]$ with x to some state $[[q_P, q', M'], q'_R]$ that is part of an included connection instance $C_{[q,x,q']}(q_R, q'_R)$. By construction, we have $[q_R, x, q'_R] \in \delta_R$. Since enabledness of a transition in $P \oplus R$ and its effect on M just depend on M and x, transition $[[q_P, q_R, M], x, [q_P, q'_R, M']]$ must be present in $P \oplus R$.

Let $[q_P, q_R, M]$ be final in $P \oplus R$, i.e. $q_P \in Q_{FP}$, $q_R \in Q_{FR}$, and $M = [\,]$. This means that, for any q, $[[q_P, q, M], q_R]$ is final in F_q and, by construction, $[[q_P, q, M], q_R]$ is final in TS_R. The other way round, a state in TS_R is final according to Definition 3.6 if q_R is final and the state of the fragment instantiated with q_R is final. By construction of fragments, this implies $q_P \in Q_{FP}$ and $M = [\,]$, so $[q_P, q_R, M]$ is final in $P \oplus R$. □

It is well known that bisimilar systems are undistinguishable for any formula of the temporal logic CTL that uses atomic propositions which are preserved by the considered bisimulation relation. Since deadlock- and livelock freedom can be expressed as *AG EF final* in this logic, Theorem 1 implies the following corollary.

Corollary 1.
$P \oplus R$ is deadlock-free and livelock-free iff TS_R is. ⌟

Consequently, verifying TS_R in place of $P \oplus R$ is fully justified.

4. Condensing fragments

In this section we aim at reducing the size of the calculated fragments and thus the number of connections. That way, we achieve the actual speed-up that we would like to have at the time of a "find" request. The reduction we are using is different from usual state space reduction known from explicit state space verification. While traditional state space reduction is applied *during* state space construction, we can apply our techniques *after* state space construction.

This is not to say that usual state space reduction techniques like the symmetry method or partial-order reduction are completely out of scope in our setting. The problem is that application of these techniques must be restricted to the interior of a fragment, since changing number or connectivity of fragments would cause trouble in the procedure of composing fragments to a transition system TS_R as described in the previous section. Consequently, we focus on an a posteriori state space reduction that can be applied immediately after having constructed the fragments. The outcome of the reduction is then available in *every* transition system that uses the reduced fragments.

4.1. SCC-based fragment reduction

Deadlocks and livelocks are terminal strongly connected components (TSCCs for short) which do not contain a final state. If this component is a singleton state without a self-loop, we have a deadlock, otherwise a livelock.

In this light, replacing all strongly connected nodes in a fragment by single nodes is an obvious reduction. It may turn a livelock into a deadlock, but the reduced system is deadlock-free and livelock-free if and only if the original system is.

Definition 4.1 (SCC reduction).
Let $\mathcal{F}$ be a set of fragments, and $\mathcal{C}$ a set of connections. The reduced sets $\mathcal{F}'$ and $\mathcal{C}'$ are defined as follows. For each fragment $F = [V_F, E_F, \Omega_F]$, let $\equiv_F$ be the equivalence on V_F where $q \equiv_F q'$ iff q and q' are mutually reachable using edges in E_F. Let F' be defined by

- $V_{F'} = V_F | \equiv_F$,
- $E_{F'} = \{[[q]_{\equiv_F}, [q']_{\equiv_F}] \mid \exists q_1 \in [q]_{\equiv_F}, q'_1 \in [q']_{\equiv_F} : [q_1, q'_1] \in E_F\}$, and
- $\Omega_{F'} = \{[q]_{\equiv_F} \mid [q]_{\equiv_F} \cap \Omega_F \neq \emptyset\}$.

Let $\mathcal{F}' = \{F' \mid F \in \mathcal{F}\}$.

For a connection C between fragments F_1 and F_2, let $[[q]_{\equiv_{F_1}}, [q']_{\equiv_{F_2}}] \in C'$ iff there exist $q_1 \in [q]_{\equiv_{F_1}}$ and $q'_1 \in [q']_{\equiv_{F_2}}$ with $[q_1, q'_1] \in C$. Let $\mathcal{C}' = \{C' \mid C \in \mathcal{C}\}$. ⌟

Since all replacements are executed simultaneously and consistently, it is easy to see that the reduction is well defined. Furthermore, all SCCs of single fragments are strongly connected in every transition system composed of the fragments. Mutual reachability of states in different SCC is left invariant. It is thus easy to see that the following corollary holds.

Corollary 2.
A transition system composed using fragments in $\mathcal{F}$ and connections in $\mathcal{C}$ is deadlock-free and livelock-free iff the corresponding reduced transition system is where, for every fragment $F \in \mathcal{F}$, F' is used instead, and every connection $C \in \mathcal{C}$ is replaced by the corresponding C'. ⌟

SCC reduction on fragments alone is applicable if the given service P has cycles in its internal activities. Since none of the fragments of Fig. 2 has an internal cycle, SCC reduction does not effect these fragments.

4.2. Rule-based reduction

Given an SCC-reduced system of fragments and connections, we can further condense these fragments and connections using local state space transformations. We show some examples for applicable rules which have been adapted from the deadlock-preserving condensation rules by Juan et al. [6].

Making the rules preserve livelocks was mostly easy as most of the rules in [6] indeed do preserve livelocks. The actual challenge in adapting the rules was rather to assure that their application is justified in *any transition system* that can be constructed using the considered fragments and connections.

For simplifying presentation, we introduce some notation. For a state q, let $F_q = [V_q, E_q, \Omega_q]$ be the fragment where $q \in V_q$. For a pair $[q, q']$, let $E_{[q,q']} = E_q$ if $F_q = F_{q'}$ and $[q, q'] \in E_q$ and let $E_{[q,q']} = C_{[F_q, F_{q'}]}$, otherwise. With this notation, we are able to refer to the fragment surrounding some state. We can further refer to transitions internal to a fragment and to transitions in connections in a single notation. For a set T of transitions, q' is reachable via T from q ($q \xrightarrow{T} *q'$) iff $[q, q']$ is in the reflexive and transitive closure of T. For a state q, let ${}^\bullet q = \{q' \mid [q', q] \text{ appears in any fragment or connection}\}$ and $q^\bullet = \{q' \mid [q, q'] \text{ appears in any fragment or connection}\}$ denote the set of predecessor and successor states of q, respectively.

We are now ready to present the actual transformation rules.

4.2.1. Rule Transitive Reduction.

The aim of the transitive reduction rule [10] is to remove a transition $[q, q']$ if and only if there is another transition sequence T from q to q'.

Constraint: There is a transition $[q, q']$ where $q \xrightarrow{(E_q \cup E_{q'} \cup E_{[q,q']}) \setminus \{[q,q']\}} *q'$.

Application: $E_{[q,q']} := E_{[q,q']} \setminus \{[q, q']\}$.

Figure 4 illustrates this rule. The following lemma proves that the above presented reduction rule is applicable in our setting.

Lemma 1.
Rule Transitive Reduction preserves deadlock- and livelock freedom in both directions. ⌟

Proof: Let TS_R be constructed of fragments and assume that it contains an instance of $[q, q']$. If this transition is internal to a fragment, then the whole sequence required in the constraint is internal to the same fragment. So q' is still reachable from q. If $[q, q']$ belongs to a connection

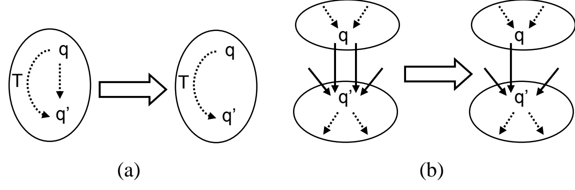


Figure 4. Examples for transitive reduction rule.

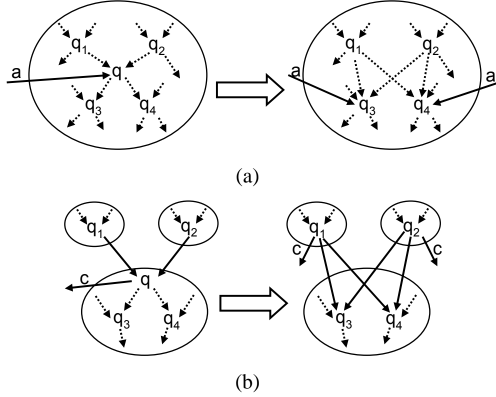


Figure 5. Examples for sequence rule.

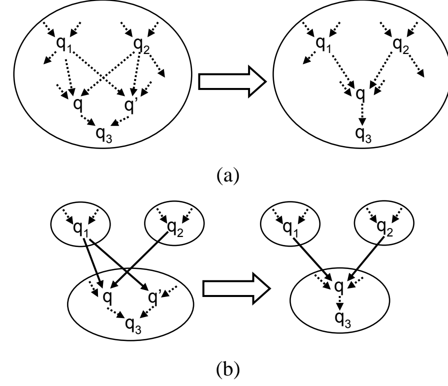


Figure 6. Examples for equivalent states rule.

$C(F_1, F_2)$, then fitting instances of both fragments and the connection are present in TS_R and the constraint again assures reachability of q' from q. Thus, mutual reachability of states is left invariant which is sufficient for asserting preservation of deadlock and livelock freedom. □

4.2.2. Rule Sequence. The intuition behind this rule is to condense sequences of states and synchronization states in order to minimize all paths to deadlock or final states. A state q can only be removed if either all predecessor or successor states of q are in the same fragment as q. Clearlx, the initial state must not be removed.

Constraint: There is a state q such that there is a transition $[q, q^*] \in E_q$ (i.e., q is not a deadlock in its own fragment); ${}^\bullet q \times \{q\} \subseteq E_q$ or $\{q\} \times q^\bullet \subseteq E_q$.

Application: For every $q' \in {}^\bullet q$ and every $q'' \in q^\bullet$, $E^* := E^* \cup \{[q', q'']\}$), where E^* is $E_{[q',q]}$ if $E_{[q',q]} \neq E_q$, and $E^* = E_{[q,q'']}$, otherwise. For all $q' \in {}^\bullet q$, $E_{[q',q]} := E_{[q',q]} \setminus \{[q', q]\}$. For all $q' \in q^\bullet$, $E_{[q,q']} := E_{[q,q']} \setminus \{[q, q']\}$. $V_q := V_q \setminus \{q\}$.

For an illustration of that rule see Fig. 5. The following lemma proves that the above presented reduction rule is applicable in our setting.

Lemma 2.
Rule Sequence preserves deadlock- and livelock freedom in both directions. ⌟

Proof: (Sketch) Let TS_R be a transition system built from fragments. Clearly, instances of manipulated connections are only present in TS_R with instances of F_q. Consider a path to a final state or a deadlock that passes through q. This path passes through some $q' \in {}^\bullet q$ and a $q'' \in q^\bullet$. By our constraint, at least one of $[q', q]$ and $[q, q'']$ is internal to the fragment of q. If one of these transitions is outside the fragment of q, the bypass transition $[q', q'']$ is member of the same connection. Thus, an instance of $[q', q'']$ is present in TS_R if and only if transitions $[q', q]$ and $[q, q'']$ are present in the unreduced system. The only removed state is q. The constraints assert that q can only be final if all predecessor or all successor states of q are final. So reachability of a final state remains invariant for all states that reach q. As q has at least one successor in its own fragment, it has a successor in TS_R. Thus, reachability of a final state from q is the same as reachability of a final state from the (still present) q^*. □

Applying this rule to the fragments of our example (cp. Fig. 2) results in removing nodes v_5 from fragment F_{s3} and v_7 and v_8 from fragment F_{s4}. Accordingly, we have to change connections $\mathsf{C}_{[v4,!c,v5]}$ and $\mathsf{C}_{[v4,!d,v7]}$ to $\mathsf{C}_{[v4,!c,v6]}$ and $\mathsf{C}_{[v4,!d,v9]}$. The example shows that this rule is a rather powerful reduction technique.

4.2.3. Rule Equivalent States. This rule aims at detecting states q and q' that share the same successors. Those states can be merged while preserving deadlock and livelock freedom.

Constraint: There are states q and q' such that $F_q = F_{q'}$, $q \in \Omega_q$ iff $q' \in \Omega_q$, and, for all fragments or connections E^*, $[q, q''] \in E^*$ iff $[q', q''] \in E^*$.

Application: For every $q^* \in {}^\bullet q'$, $E_{[q^*,q']} := (E_{[q^*,q']} \setminus \{[q^*, q']\}) \cup \{[q^*, q]\}$; For every $q'' \in q'^\bullet$, $E_{[q',q'']} := E_{[q',q'']} \setminus \{[q', q'']\}$; $V_q := V_q \setminus \{q'\}$.

For an illustration of that rule see Fig. 6. The following lemma proves that the above presented reduction rule is applicable in our setting.

Lemma 3.
Rule Equivalent States preserves deadlock- and livelock freedom in both directions. ⌟

Proof: (Sketch) Let TS_R be a transition system built from fragments and consider an instance of F_q. This fragment contains both q and q' in its unreduced version. Since every transition from q is doubled by a transition from q' to the same state in the same fragment or connection, redirection of incoming transitions to q instead of q' does not change reachability of final states. The redirected transitions appear in the same fragment or connection as the original one, so the substitution is independent of the actual construction of TS_R. □

From Lemma 1–3 presented above we can conclude the following corollary.

Corollary 3.
Let TS'_R result from applying any sequence of the reduction rules as described above to TS_R. Then, TS_R has a deadlock (livelock) iff TS'_R has a deadlock (livelock). ⌟

The corollary states that we can condense fragments by applying the three reduction rules in any order and the minimized fragments preserve deadlocks and livelocks.

5. Case study

The results presented in this paper have been prototypically implemented in our service analysis tool Fiona[1] [11]. Among others Fiona can be used to read two service models P and R, calculate the most permissive strategy $MP(P)$ for P together with the (minimized) fragments and connections of P, and construct the state space $P \oplus R$ from these fragments. Our service models are *open nets* [2] — Place/Transition Petri nets extended by an interface. For the analysis, Fiona computes the reachability graph of the open net (i.e., a labeled transition system).

Table 1 provides the results of our case study including 16 services (specified as open nets). The first six examples are realistic services taken from the WS-BPEL specification [3] ("Loan Approval", "Purchase Order" and "Travel Service 1"), and from [12] ("Olive Oil Ordering"). "Travel Service 2" is a modification of "Travel Service 1". As these examples were specified in the service description language WS-BPEL [3], we had to translate them into open nets using the compiler BPEL2oWFN [13].

The "Beverage Machine" is taken from [2]; the online shops are taken from [13]. "Philosophers" are an open net model of three and five dining philosophers. "SMTP Protocol" models the SMTP protocol. The last five processes models are real-life service models provided by a consultant company.

For each service P, Table 1 provides information about the state space of P, the state space of the most permissive strategy $MP(P)$ for P, the size of the fragments, and the time for calculating $MP(P)$ and computing the reduced fragments. More precisely, $|Q_P|$, $|L_P|$, and $|\delta_P|$ refer to the number of states, interface channels, and transitions in P, respectively; $|Q|$ and $|\delta|$ refer to the number of states and edges in $MP(P)$, respectively; $|V|$ and $|V_{red}|$ refer to the number of states of *all* fragments before and after applying the reduction rules. $\frac{|V|}{Q}$ and $\frac{|V_{red}|}{Q}$ show the average number of states *per* fragment before and after applying the reduction rules. Finally, $t_{MP(P)}$ denotes the time for computing the most permissive strategy for P, and t_{red} shows the time needed for reducing the fragments.

As an example, the most permissive strategy of "Online Shop 1" has 12 states. Thus, Fiona computed 12 fragments. The sum of all states of these 12 fragments is $|V| = 137$. Applying the proposed abstraction rules results in $|V_{red}| = 15$ states. So we have in average 11.4 states per fragment before and 1.3 states per fragment after the reduction. Hence, at a "find" request only $\frac{15}{137} = 11\%$ of the actual state space has to be model checked. The time for computing $MP(P)$ is six seconds; reducing the fragments takes no time.

Based on the case study we make the following three observations.

The average number of states per fragment is not very high in general—for example, in nine service models lower than five. Consequently, there is little scope for reduction in such examples. One reason might be that all open nets have been structurally reduced using the well-known Murata rules [14] before transforming their state spaces. We apply these rules, because they significantly speed-up the computation of the most permissive strategy while preserving all relevant properties.

For three examples, the overall computation time takes more than an hour. This reflects the high worst case complexity of the algorithm for calculating the most permissive strategy [9]. The algorithm computes an overapproximation of the possible states of the most permissive strategy and then iteratively removes all states which cause violations of the properties to be preserved. As an example, for computing the most permissive strategy for "Process 4" more than 400,000 states had to be removed. Also the minimization of the fragments takes in four examples more than 10 minutes. The reason is that the transitive reduction rule has a high worst-case complexity [10], and we were interested in the best possible reduction. So there is obviously a trade-off between fragment size and run time that could be tackled—for example, by limiting the number of iterations through the state space to detect nodes and edges that can be removed.

The proposed abstraction rules seem to be powerful as the state spaces of the fragments are reduced significantly. None of the reduced example processes has more than 2.5 states per fragment in average, and five processes contain only a single state per each fragment. Consequently, building the transition system TS_R for $P \oplus R$ from reduced

1. available at http://www.service-technology.org/fiona

Table 1. Experimental results running Fiona. All experiments were taken on an UltraSPARC III processor with 900MHz and 4 GB RAM running Solaris 10.

	P			$MP(P)$		$\mathcal{F}(P)$				time (h:min:sec)	
Service	$\|Q_P\|$	$\|L_P\|$	$\|\delta_P\|$	$\|Q\|$	$\|\delta\|$	$\|V\|$	$\|V_{red}\|$	$\frac{\|V\|}{Q}$	$\frac{\|V_{red}\|}{Q}$	$t_{MP(P)}$	t_{red}
Loan Approval	26	6	33	7	8	43	10	6.1	1.4	0:00:01	0:00:00
Purchase Order	12	10	15	168	548	464	168	2.8	1.0	0:00:03	0:00:04
Olive Oil Ordering	6	6	6	16	27	33	20	2.1	1.3	0:00:00	0:00:00
Travel Service 1	7	8	7	56	140	120	56	2.1	1.0	0:00:00	0:00:01
Travel Service 2	10	12	11	288	1,008	672	320	2.3	1.1	0:00:22	0:00:10
Online Shop 1	205	7	463	12	15	137	15	11.4	1.3	0:00:04	0:00:00
Online Shop 2	308	8	744	7	7	77	8	11.0	1.1	0:00:06	0:00:00
Beverage Machine	5	7	8	11	24	37	15	3.4	1.4	0:00:00	0:00:01
Philosophers #3	46	6	70	67	96	499	67	7.5	1.0	0:00:01	0:00:01
Philosophers #5	574	10	1,476	1,432	3,435	43,848	1,432	30.6	1.0	0:05:06	0:39:19
SMPT Protocol	8,345	12	34,941	1,216	4,752	13,148	2,894	10.8	2.4	4:10:05	0:29:10
Registration	1,057	6	2,927	7	8	2,239	13	320.0	1.9	0:00:20	0:00:32
Process 1	97,511	20	468,072	896	3,924	1,428	1,365	1.6	1.5	0:00:10	0:02:41
Process 2	244	13	758	1,607	7,206	4,844	3,782	3.0	2.4	0:00:12	0:16:12
Process 3	992	13	3744	236	724	1,171	236	5.0	1.0	1:14:36	0:00:12
Process 4	160	19	494	6,820	40,350	24,688	13,903	3.6	2.0	4:41:06	6:07:55

fragments results in all examples in a *smaller* state space (compared to the transition system composed from non-reduced fragments). We assume that model checking those reduced state spaces result in a speed-up due to the smaller number of states and hence justifies our proposal for shifting computation time from the time-critical find phase to the publish phase. A validation of this assumption is, however, subject of ongoing research.

6. Related Work

There is a lot of research being done to provide a methodology to support matchmaking in service-oriented architectures. As already mentioned in the introduction, some approaches propose to compute and publish a public view P' of a provided service P [15], [16]. Then, upon a "find" request of a service R matchmaking means model checking of the composition $R \oplus P'$ to decide proper interaction. However, in these approaches the computational effort is during every "find" request whereas our approach shifts these effort to the "publish" phase.

Other approaches such as [1], [2] reduce the implementation of a "find" request to a matching problem. However, in these approaches proper termination is restricted to deadlock freedom whereas we also consider livelock freedom.

The work presented in this paper is related to work on minimization of transition systems (see [17]–[19], for instance). These papers present congruences w.r.t. composition which never equate a transition system with a property such as deadlock or livelock freedom with one which does not satisfy that property. That way, one can minimize a service P to a service P' that preserves deadlocks and livelocks. However, as R is unknown, at publish phase only P can be minimized whereas in our approach the state space of $P \oplus R$ can be minimized.

Juan et al. [6] define several deadlock-preserving reduction rules for state machines modeling reactive systems. In Sect. 4, we have generalized these rules and adapted them to preserve livelocks as well.

7. Conclusion

We proposed a methodology to construct the state space $P \oplus R$ of finite state machines P and R from a finite set of (state space) fragments of P. A fragment contains that part of $P \oplus R$ that takes place between two subsequent transitions of R. That way, $P \oplus R$ can be computed by composing fragments of P according to the behavior of R. We further presented several reduction rules to minimize the state space of the fragments.

Our work is motivated by a scenario in service-oriented architectures where services are published in a registry and have to be selected upon a service request. That means, given a service R, one has to select a service P from a registry and to check whether the composition $P \oplus R$ is free of deadlocks and livelocks. Since the number of "find" requests is much greater than the number of "publish" events, we proposed to publish (reduced) fragments of P. That way, computational effort in reducing the state space $P \oplus R$ is shifted from "find" to "publish", and thus a "find" request is reduced to model check the reduced state space of $P \oplus R$.

A prototype implementation experimented with a number of real-life service models. The results obtained have proven that our reduction techniques reduce significantly the state space of fragments. Thus, the computational effort to model check the resulting state space $P \oplus R$ for the absence of deadlocks and livelocks is drastically reduced in general.

In ongoing work we plan to continue validating our approach. In addition, we will work on a procedure to decide whether or not every strategy for a service P is a strategy for

service P' given their sets of fragments. That way, we can decide whether P can be substituted by P' without affecting any client of P.

Acknowledgment

The authors would like to thank Daniela Weinberg for her work on the implementation of Fiona and Dirk Fahland and Niels Lohmann for their help on the case study. Christian Stahl has been funded by the DFG within grant "Substitutability of Services" (RE 834/16-1). Karsten Wolf was supported by the DFG within grant "Operating Guidelines for Services" (WO 1466/8-1).

References

[1] A. Wombacher, B. Mahleko, and E. J. Neuhold, "IPSI-PF - A business process matchmaking engine based on annotated finite state automata," *Inf. Syst. E-Business Management*, vol. 3, no. 2, pp. 127–150, 2005.

[2] N. Lohmann, P. Massuthe, and K. Wolf, "Operating Guidelines for Finite-State Services," in *ICATPN 2007*, ser. LNCS, J. Kleijn and A. Yakovlev, Eds., vol. 4546. Springer, 2007, pp. 321–341.

[3] A. Alves et al., "Web Services Business Process Execution Language Version 2.0," OASIS, OASIS Standard, 11 April 2007, Apr. 2007.

[4] N. Lohmann, O. Kopp, F. Leymann, and W. Reisig, "Analyzing BPEL4Chor: Verification and Participant Synthesis," in *WS-FM 2007*, ser. LNCS, M. Dumas and R. Heckel, Eds., vol. 4937. Springer, 2008, pp. 46–60.

[5] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, Massachusetts: MIT Press, Jan. 2000.

[6] E. Y. T. Juan, J. J. P. Tsai, and T. Murata, "Compositional Verification of Concurrent Systems Using Petri-Net-Based Condensation Rules." *ACM Trans. on Programming Languages and Systems*, vol. 20, no. 5, pp. 917–979, 1998.

[7] R. Milner, *Communication and Concurrency*. Prentice-Hall, Inc., 1989.

[8] D. Park, "Concurrency and Automata on Infinite Sequences," in *Theoretical Computer Science*, ser. LNCS, P. Deussen, Ed., vol. 104. Springer, 1981, pp. 167–183.

[9] K. Wolf, "Does my service have partners?" *LNCS ToPNoC II*, vol. 5460, pp. 152–171, Mar. 2009.

[10] A. V. Aho, M. R. Garey, and J. D. Ullman, "The transitive reduction of a directed graph," *SIAM J. Comput.*, vol. 1, no. 2, pp. 131–137, 1972.

[11] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg, "Analyzing interacting WS-BPEL processes using flexible model generation," *Data Knowledge Engineering*, vol. 64, no. 1, pp. 38–54, 2008.

[12] J. Arias-Fisteus, L. S. Fernández, and C. D. Kloos, "Applying model checking to BPEL4WS business collaborations," in *SAC 2005*, H. Haddad, L. M. Liebrock, A. Omicini, and R. L. Wainwright, Eds. ACM, 2005, pp. 826–830.

[13] N. Lohmann, P. Massuthe, C. Stahl, and D. Weinberg, "Analyzing Interacting BPEL Processes," in *BPM 2006*, ser. LNCS, S. Dustdar, J. L. Fiadeiro, and A. Sheth, Eds., vol. 4102. Springer, Sep. 2006, pp. 17–32.

[14] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[15] W. M. P. v. d. Aalst and M. Weske, "The P2P approach to Interorganizational Workflows," in *CAiSE 2001*, ser. LNCS, K. R. Dittrich, A. Geppert, and M. C. Norrie, Eds., vol. 2068. Springer, 2001, pp. 140–156.

[16] F. Leymann, D. Roller, and M.-T. Schmidt, "Web services and business process management." *IBM Systems Journal*, vol. 41, no. 2, pp. 198–211, 2002.

[17] A. Valmari, "The Weakest Deadlock-Preserving Congruence." *Inf. Process. Lett.*, vol. 53, no. 6, pp. 341–346, 1995.

[18] A. Puhakka and A. Valmari, "Weakest-Congruence Results for Livelock-Preserving Equivalences." in *CONCUR 1999*, ser. LNCS, J. C. M. Baeten and S. Mauw, Eds., vol. 1664. Springer, 1999, pp. 510–524.

[19] J.-C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu, "CADP - A Protocol Validation and Verification Toolbox," in *CAV 1996*, ser. LNCS, R. Alur and T. A. Henzinger, Eds., vol. 1102. Springer, 1996, pp. 437–440.