Wendy: A tool to synthesize partners for services[*]

Niels Lohmann
Universität Rostock, Institut für Informatik

Daniela Weinberg
Universität Rostock, Institut für Informatik

Abstract. Service-oriented computing proposes *services* as building blocks which can be composed to complex systems. To reason about the correctness of a service, its communication protocol needs to be analyzed. A fundamental correctness criterion for a service is the existence of a *partner service*, formalized in the notion of *controllability*.

In this paper, we introduce *Wendy*, a Petri net-based tool to synthesize partner services. These partners are valuable artifacts to support the design, validation, verification, and adaptation of services. Furthermore, Wendy can calculate an *operating guideline*, a characterization of the set of all partners of a service. Operating guidelines can be used in many application scenarios from service brokerage to test case generation. Case studies show that Wendy efficiently performs on industrial service models.

1. Objectives

In the emerging field of service-oriented computing (SOC) [28] it is proposed to build complex systems by composing geographically and logically distributed services. A service encapsulates a certain functionality and offers it through a well-defined interface. Conceptually, SOC revives old ideas from component-based design [27, 32] or from programming-in-the-large [8], for instance.

A simple realization of SOC is the encapsulation of classical computer programs which calculate an output from given inputs as *remote procedure calls* or *stateless services*. Such services only exchange pairs of request/response messages and are capable of implementing simple systems such as stock or weather information systems. This approach is insufficient to implement real-world business scenarios,

Address for correspondence: Niels Lohmann, Universität Rostock, Institut für Informatik, 18051 Rostock, Germany, niels.lohmann@uni-rostock.de

[*]Extended abstract appeared as [22].

which do not only calculate an output from given inputs, but in which messages are constantly sent back and forth. Examples for such *stateful* conversations are price negotiations, auctioning, or scenarios in which exception handling is necessary. In this setting, more complex interactions need to be considered and a service needs to implement a *communication protocol* (also called *business protocol* [29]) which specifies the order in which the service's activities are executed and which may distinguish arbitrary states of the interaction with other services. The most prominent class of services are *Web services* [4]. Here, the Internet and several Web-related standards are used to realize SOC. This makes services virtually independent of their geographical location and technological context and allows to entirely focus on the functions a service offers.

To embrace this communicating nature of services, *controllability* [35] has been introduced as a fundamental correctness criterion for services. A service is controllable iff there exists a *partner service* such that their composition is compatible, for instance free of deadlocks. Controllability is not only a fundamental correctness criterion for a service — without a compatible partner service, a service is essentially useless —, but a partner service also provides valuable insight into the communication behavior of the modeled service. Furthermore, it is a useful artifact to validate [19] or document the service and is the basis of adapter synthesis [9]. In recent work [3], partner services are also used to automatically configure business processes.

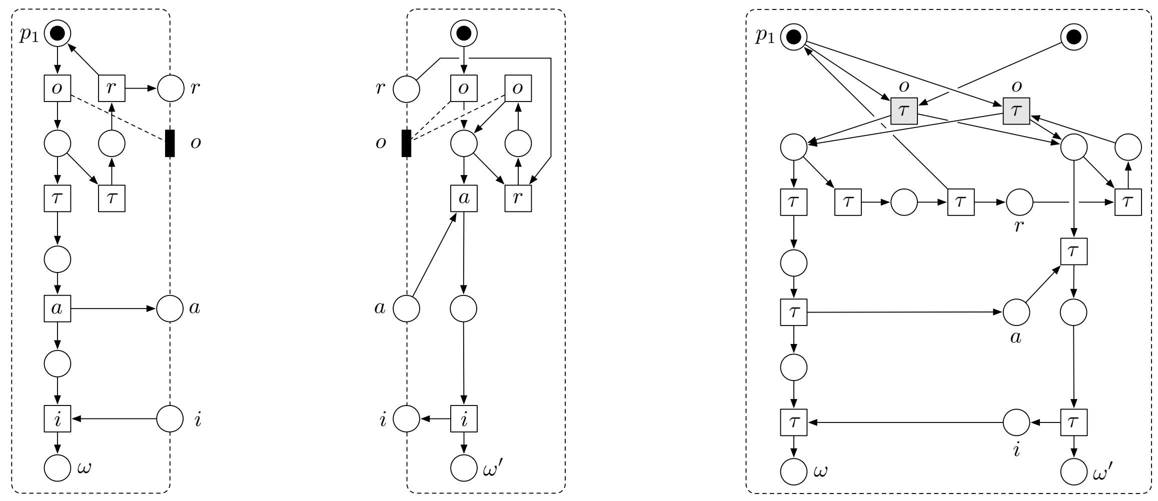
A controllable service usually has more than one partner service. An *operating guideline* finitely characterizes the (possibly infinite) set of all partner services of a service [20]. Operating guidelines are useful in a variety of applications including test case generation [10], service correction [13], instance migration [11], or service substitution [31]. They further allow to efficiently realize a service-oriented architecture in which the service provider publishes his operating guideline at a service broker. A service requester then only needs to check whether his service is one of the partner services which is characterized by the operating guideline [18].

In this paper, we introduce Wendy, a tool to synthesize partner services and to calculate the operating guideline of a service. Wendy provides the basis of a vast variety of applications which are essential in the paradigm of SOC. Case studies show that Wendy can cope with industrial and academic service models. We continue with sketching the functionality and the used formalism. The architecture of Wendy and the components it is built of are described in Sect. 4 together with some heuristics that aim at reducing Wendy's time and memory consumption. Section 5 shows how Wendy can be used in different use cases and how it is integrated into other tools, provides experimental results, gives information about how to obtain Wendy, and discusses improvements with respect to a previous implementation. Section 6 concludes the paper.

2. Background and setting

The theory [20, 35, 33] implemented by Wendy focuses on the behavior (both control flow and communication protocol) of a service. We model a service as an *open net* [24], a special type of Petri net with an interface. With open nets, we can faithfully model both synchronous and asynchronous communication, concurrent behavior, and we can easily compose open nets by merging interfaces. Open nets can be automatically derived from industrial service description languages such as WS-BPEL [14].

To introduce open nets, consider the example in Fig. 1(a). The net N_{buyer} models a buyer service that receives offers (o) from a client and decides whether to accept (a) or to reject (r) the offer. This decision



(a) seller service N_{sell} (b) buyer service N_{buy} (c) composition $N_{\text{sell}} \oplus N_{\text{buy}}$

Figure 1. A seller (a) and a buyer service (b) modeled as open nets N_{sell} and N_{buy}, and their composition (c).

is modeled by internal τ-steps and is nondeterministic. In case the offer is rejected, the service returns to its initial marking $[p_1]$ (one token on place p_1) and waits for another offer. In case the offer is accepted, the service eventually receives an invoice (i) and reaches the marking $[\omega]$. The interfaces (modeling the message channels of the service) are depicted on the dashed frames. We distinguish asynchronous input (i) and output (a, r) message channels (modeled as places) and synchronous message channels (o), depicted as black rectangles. To differentiate desired *final markings* from unwanted deadlocks, an open net has a distinguished final marking ($[\omega]$ for N_{buyer}). We require the interface places to be empty in the initial and final marking. Formally, an open net is defined as follows.

Definition 2.1. (Open net)
An *open net* $N = [P, T, F, m_0, \Omega, I, O, S, \ell]$ consists of a Petri net $[P, T, F, m_0]$ (called the inner of N), a set of final markings Ω, a set of asynchronous input channels I, asynchronous output channels O, synchronous channels S, and a labeling function $\ell : T \to (I \cup O \cup S \cup \{\tau\})$. We explicitly model asynchronous channels by places; that is, $I, O \subseteq P$ such that $p \in I$ implies ${}^\bullet p = \emptyset$ and $p \in O$ implies $p^\bullet = \emptyset$. Further, for $p \in (I \cup O)$ let $m_0(p) = 0$ and $m_f(p) = 0$ for all $m_f \in \Omega$. We call an open net *closed* if $I = O = S = \emptyset$.

The open nets N_{buyer} and N_{seller} (modeling a selling service, cf. Fig. 1(b)) can be *composed* by merging the input places of N_{buyer} with the output places of N_{seller} (and vice versa), and by fusing each pair of transitions of N_{buyer} and N_{seller} which are connected to the same synchronous channel (depicted gray in the composition $N_{\text{buyer}} \oplus N_{\text{seller}}$ in Fig. 1(c)). Initial and final markings are added element-wise:

Definition 2.2. (Composition)
For $i \in \{1, 2\}$, let $N_i = [P_i, T_i, F_i, m_{0_i}, \Omega_i, I_i, O_i, S_i, \ell_i]$ be open nets. N_1 and N_2 are *composable* iff $I_1 = O_2$, $O_1 = I_2$, and $S_1 = S_2$. The composition of two composable nets N_1 and N_2 (denoted $N_1 \oplus N_2$) is a closed net $N = [P, T, F, m_0, \Omega, I, O, S, \ell]$ with

- $P = P_1 \cup P_2$,

- $T = ((T_1 \cup T_2) \setminus \{t \mid t \in (T_1 \cup T_2) \wedge \ell(t) \in S\}) \cup \{[t_1, t_2] \mid t_1 \in T_1 \wedge t_2 \in T_2 \wedge \ell(t_1) = \ell(t_2) \wedge \ell(t_1) \in S\}$,
- $F = ((F_1 \cup F_2) \cap ((P \times T) \cup (T \times P))) \cup \{[p, [t_1, t_2]] \mid [p, t_1] \in F_1 \vee [p, t_2] \in F_2\} \cup \{[[t_1, t_2], p] \mid [t_1, p] \in F_1 \vee [t_2, p] \in F_2\}$,
- $m_0 = m_{0_1} \oplus m_{0_2}$, $\Omega = \{m_{f_1} \oplus m_{f_2} \mid m_{f_1} \in \Omega_1 \wedge m_{f_2} \in \Omega_2\}$,
- $I = S = O = \emptyset$, and $\ell(t) = \tau$ for all $t \in T$.

Thereby, the composition of two markings m_1 and m_2 is defined as $(m_1 \oplus m_2)(p) = m_1(p)$ iff $p \in P_1$ and $(m_1 \oplus m_2)(p) = m_2(p)$ iff $p \in P_2$.

The composition $N_{\text{buyer}} \oplus N_{\text{seller}}$ is *compatible*: the only reachable deadlock $[\omega, \omega']$ is a final marking and the places modeling the asynchronous interface are bounded. If two open nets N and N' are compatible, we call both N and N' *controllable* [35], and refer to N as a partner service of N', and vice versa.

Definition 2.3. (Compatibility, controllability)
A closed net is *compatible* iff (1) every previous asynchronous communication place is bounded and (2) for each reachable marking m with $m_0 \xrightarrow{*} m$ holds: if m does not enable a transition, then $m \in \Omega$. An open net N is *controllable* iff there exists an open net M such that $N \oplus M$ is compatible.

Compatibility is a fundamental correctness criterion for closed service compositions. We are aware of more sophisticated criteria, for instance livelock freedom, exclusion of dead activities, or satisfaction of certain temporal logic formulae. Nevertheless, deadlock freedom and bounded communication would be certainly part of any refined correctness notion. Controllability allows us to reason about a single service while taking its communicational behavior (i.e., the service's local contribution to overall compatibility) into account. It is a fundamental correctness criterion for open services, because a service that cannot interact deadlock and bounded freely with *any* other service is certainly ill-designed. From a practical point of view, a partner service of a service N does not only prove its controllability, but is also a valuable tool to validate, test, or document the service N. Furthermore, a synthesized strategy can be used as a *communication proxy*, which can be implemented (i.e., refined) toward an executable service that is by design compatible to the original service.

Whereas checking compatibility of two given open nets is a standard model checking problem, finding a partner service is a *synthesis* problem. Wendy is a tool to synthesize partner services and the subsequent sections describe its functionality.

3. Functionality

3.1. Partner synthesis

Wendy analyzes controllability of an open net and synthesizes a partner as a witness if the net is controllable. This partner is an automaton model which can be transformed into an open net using known tools such as Petrify [6]. In the automaton representation, asynchronous send actions, asynchronous receive actions, and synchronous actions are preceded by "!", "?", and "#", respectively.

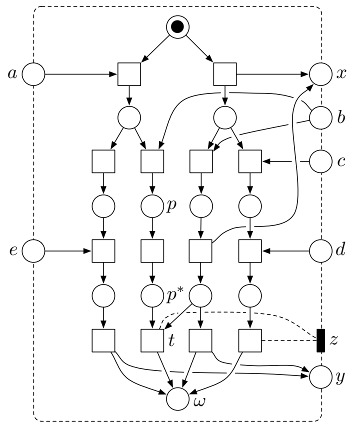


Figure 2. Example open net N.

To illustrate the functionality of Wendy, we use a slightly more complicated open net N, depicted in Fig. 2). Compared to the example of the pervious section, it has a larger interface and exhibits concurrency.

The open net N is controllable and Fig. 3(a) shows the partner synthesized by Wendy. By design, this partner is *most-permissive* in the sense that it simulates any other partner including N'. Therefore, this partner reveals that no compatible partner of N will ever send an a-message. The transition connected to the input channel a and the transition connected with channel x are in conflict. It can never be ensured that N will always first receive an a-message whenever it is available. It may as well send an x-message which leads the net into the right hand branch where no a-message will ever be received. Consequently, the a-message remains on the message channel and the final marking becomes unreachable.

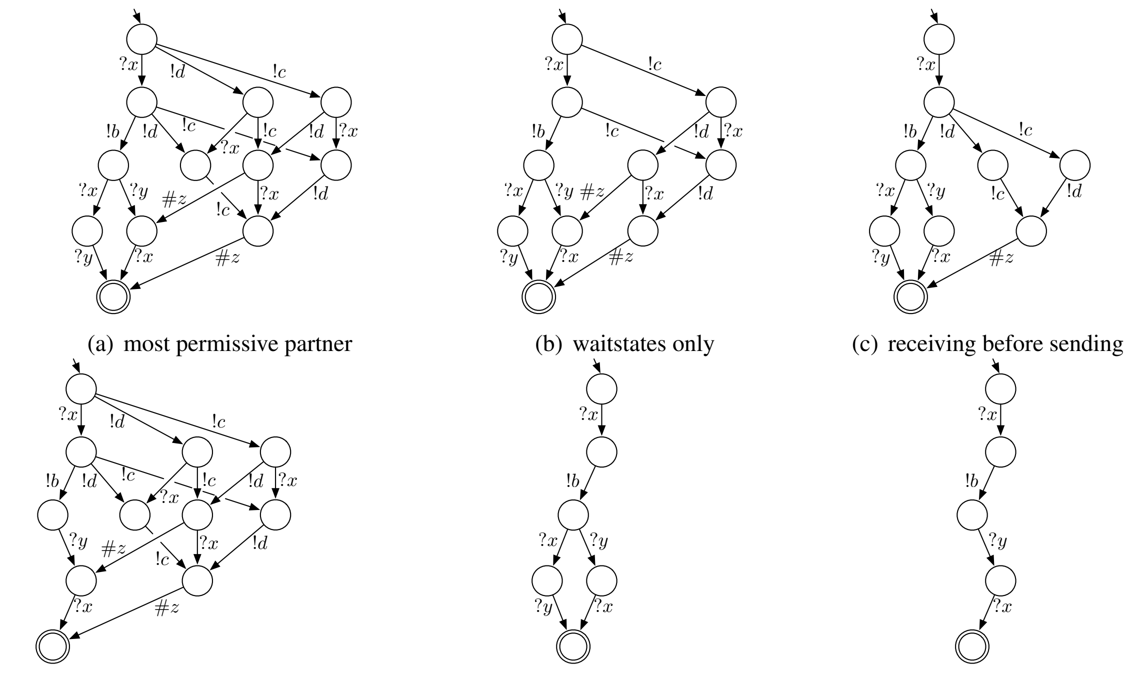
The validation of N can be refined by applying behavioral constraints [19] which filter the set of partners, for instance to only those partners sending a b-message.

3.2. Operating guidelines

Although every partner of N is simulated by the most-permissive partner, the converse does not hold: Simulation does not take full branching behavior into account and would also consider "incomplete" partners which do not implement all possible outcomes of a service's choice. To characterize exactly the set of all partners of N, we annotate the states of the most-permissive partner with Boolean formulae (see [20] for details). This annotated most-permissive partner is called an *operating guideline*. Wendy can calculate an operating guideline by generating the Boolean formulae in a postprocessing step.

3.3. Types of partners

Wendy synthesizes different types of partners depending on the analysis goal: (1) it synthesizes a most permissive partner which serves as the basis for the calculation of the operating guideline; (2) for checking controllability, the type of the synthesized partner is not of much interest. Here, we focus on a quick answer about the mere existence of a partner. To this end, a set of partner synthesis rules can be applied during the partner generation which influence the behavior and, thus, the structure of the partner; (3) by



(a) most permissive partner (b) waitstates only (c) receiving before sending

(d) sequentialize receiving events (e) succeeding sending event (f) quit as soon as possible

Figure 3. Different partners of N by applying a certain partner synthesis rule.

combining the partner synthesis rules in a certain way, Wendy generates a particular partner which will be used later on, for instance for generating an adapter service [9].

The first three partner synthesis rules (cf. Fig. 3(b)–(d)) focus on reducing the overhead due to asynchronous communication, whereas the last two rules (cf. Fig. 3(e)–(f)) always lead to small partners and hence a quick answer with respect to controllability. See [33] for further information on the partner synthesis rules.

- *Waitstates only (WSO).* Send a message or synchronize only if it is necessary for N to move on. Messages are not sent in advance.
- *Receiving before sending (RBS).* Before sending a message or synchronization, receive every asynchronous message sent by N.
- *Sequentialize receiving events (SRE).* Receive the messages sent by N in a certain order again. This, however, does not necessarily have to be the order the messages have been sent by N.
- *Succeeding sending event (SSE).* Quit any interaction as soon as one sent message leads to a proper interaction with N.
- *Quit as soon as possible (QSP).* Quit the interaction if any (synchronous or asynchronous) action has led the interaction with N to a proper end.

The partner synthesis rules may be combined arbitrarily. However, only a few combinations are reasonable, whereas other combinations are less appropriate. In the following, we list a few types of

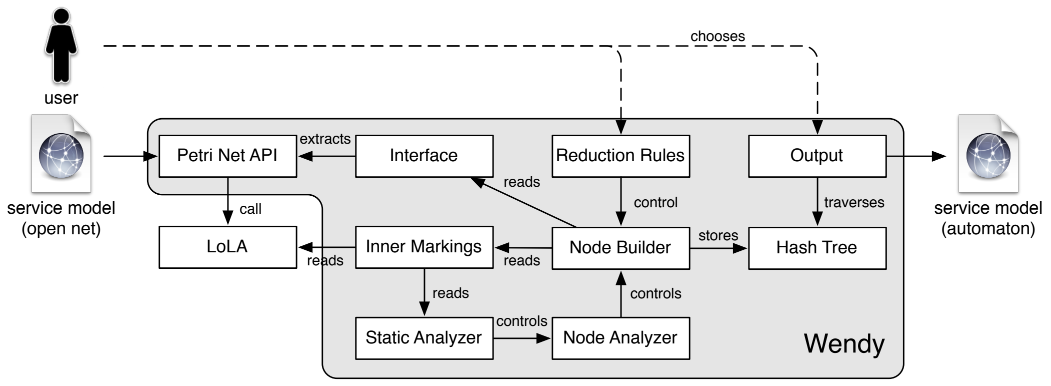


Figure 4. Architecture overview.

partners which can be synthesized by combining different partner synthesis rules (shown in brackets at the end of each description).

- *Chatty partners* send as much as possible and receive messages consecutively (SRE). These partners are best suited for adapter synthesis [9], because they hardly block the overall system, but send messages as early as possible.

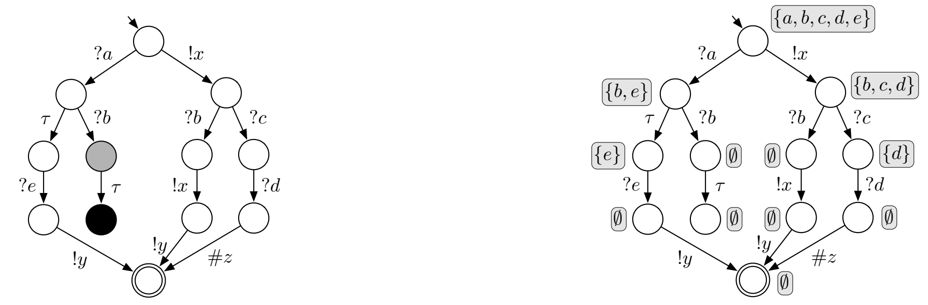
- *Arrogant partners* let service N do all the work for proper interaction and only react if it is necessary (WSO, RBS, SRE). The generated partner shows which messages of N can be received after an interaction between the partner and N has taken place. The interaction is mostly guided by N, because the generated partner does not send any messages in advance. As the messages sent by N are received sequentially, these partners tend to have a less complex structure in case N sends out many messages.

- *No-talker partners* only react to the actions of service N and quit as soon as one sent message leads to a proper end (WSO, RBS, SSE). In contrast to the arrogant partners, the behavior of these partners also reflect which messages of N may be received in parallel and may thus be pending on the message channels at one time.

- *Lazy partners* do not like to interact with N (WSO, RBS, SRE, QSP). These partners are the best choice for a quick answer on whether service N is controllable or not. Our case studies show, that even for very complex real life services, Wendy could always generate such a lazy partner within a few seconds.

4. Architecture and heuristics

Wendy is written in C++ and built up out of several components. Figure 4 sketches the overall architecture. To process the input open net, given in a file format as sketched in Fig. 7(a) or as a PNML extension (Fig. 7(b)), Wendy uses the Petri Net API [21], a C++ library encapsulating Petri net-related functions. It extracts the interface of the open net and calls LoLA [34] as an external tool to generate the reachability



(a) early deadlock detection (b) smart sending events

Figure 5. Annotated inner of the open net N with heuristic annotations.

graph of the *inner* of the open net (i. e., the open net without its interface). Figure 5(a) depicts this graph for the running example.

In contrast to an on-the-fly approach, all reachable markings are generated before any calculation is done. This approach enables Wendy to statically analyze these *inner markings* which leads to a great improvement of the overall calculation time. To synthesize a partner, the node builder calculates an overapproximation (see [35] for details) of a possible partner, and the node analyzer removes "bad" nodes. Bad nodes represent situations in which the net to be controlled reaches a deadlock, a livelock, or in which a certain number of pending messages is exceeded. With the help of the information of the static analyzer, "bad" nodes can be detected at an early stage by just considering the involved inner markings. This allows to prematurely end the generation of new nodes and to speed up the partner synthesis. In the following, we shall describe two of the implemented static analyses.

4.1. Early deadlock detection

First, deadlocks can be detected as early as possible. For instance, open net N contains the deadlock $[p^*]$ of the inner (depicted black in Fig. 5(a)), because transition t is dead. With the help of the information of the static analyzer, every node containing a deadlock — such as marking $[p^*]$ in N — is declared "bad" right away, and no more successor nodes of this node are calculated, because such inner deadlocks cannot be resolved by sending or receiving messages. Additionally, inner markings from which a deadlock cannot be avoided any more (e. g., marking $[p]$ in N — depicted gray in Fig. 5(a)) are treated similarly. This *early deadlock detection* has a great impact on the runtime in case an open net contains several deadlocks, for instance due to the application of behavioral constraints [19]. The same technique is likewise applicable to the early detection of livelocks.

4.2. Smart sending

Asynchronous message exchange allows messages to keep pending on their channel until they are finally received by the open net. Consequently, the synthesis algorithm calculates, for each state, all possible successor states that are reachable by sending a message. Hence, the second reason for a net not reaching a final marking are pending messages that will never be received. The question whether or not a message can actually be received by the net in the future can again be answered by analyzing the reachability graph of the inner of the net. We therefore check, for each state, which receiving events are reachable in the future. As a consequence, only these events need to be considered when synthesizing a partner service.

For our running example, these sets of events are written next to the states of Fig. 5(b). We see that even for this small example, these sets are small or even empty most of the time — an observation that even holds for large industrial examples. This heuristic does not restrict the set of partners in case livelock freedom is chosen as correctness criterion. In case of deadlock freedom, messages may never be received when the service has reached a livelock. In this case, the heuristic may be incomplete, but the partners that are not synthesized do not model intended behavior from a practical point of view where sent messages should be eventually received.

It is worthwhile mentioning that both checks can be realized in a single depth-first traversal of the reachability graph that is passed from LoLA to Wendy. Furthermore, a lot of information can be removed from memory after the static analysis step. For instance, the concrete distribution of tokens can be abstracted away — only the fact whether a marking is final or an (unavoidable) deadlock/livelock needs to be stored. Similarly, the net's transitions need not to be identified, but only the name of the message event (e.g., $?a$, $!y$, $\#z$, or τ) is relevant.

In addition to the node analyzer, the chosen partner synthesis rules influence the synthesis of the partner as well. The calculated nodes are compactly stored in a hash tree to quickly detect already calculated nodes. Finally, the synthesized partner is returned as either an automaton or an operating guideline.

In conclusion, the core design goals of Wendy are to (1) decrease the runtime by gathering as much information about the service model as possible during preprocessing (e. g., by analyzing the inner markings); (2) decrease the memory consumption needed to synthesize partners by implementing very problem-specific abstract data types and by keeping as little information as possible in memory.

5. Using Wendy

5.1. Use cases

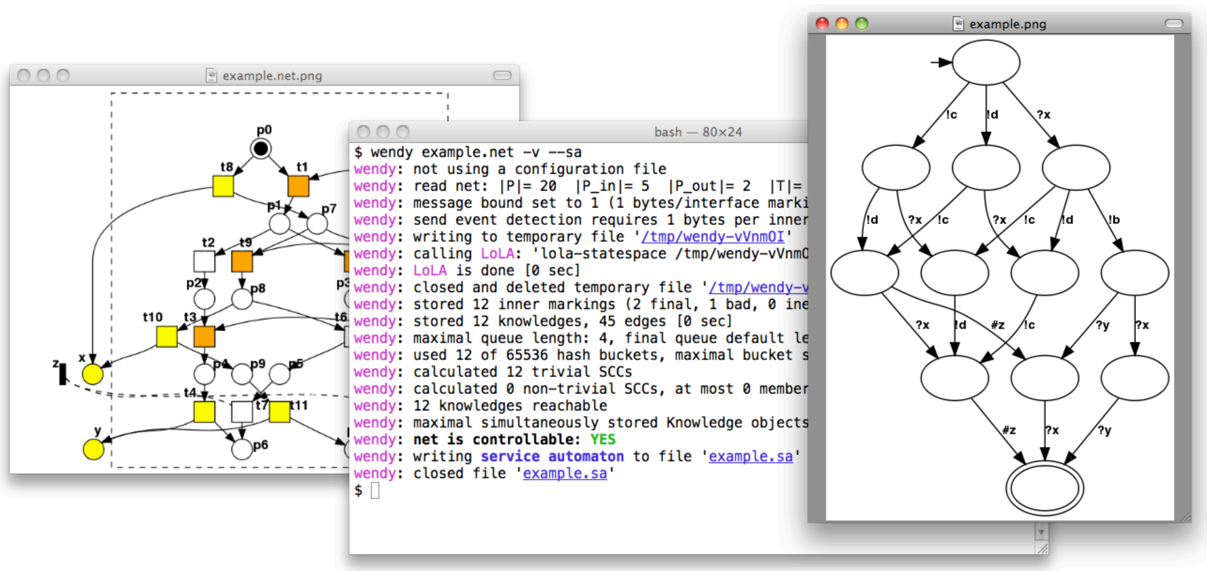
Wendy is a command-line tool implementing the following use cases. We assume an open net is given as file "service.net" in the format sketched in Fig. 7(a). Alternatively, Wendy can also process PNML files with an extension to model interfaces and final markings, cf. Fig. 7(b).

- *Partner synthesis.* To check if the open net is controllable and to synthesize a most permissive partner if one exists, call Wendy with `wendy service.net --sa`.

- *Operating guidelines.* By invoking Wendy with the following command, the operating guideline of the given open net is calculated: `wendy service.net --og`.

- *Partner synthesis rules.* To synthesize a *no-talker partner* by combining partner synthesis rules *waitstates only*, *receiving before sending* and *succeeding sending event*, call Wendy with
```
wendy service.net --sa --waitstatesOnly --receivingBeforeSending
--succeedingSendingEvent.
```

In all three use cases, Wendy will print out whether the given open net is controllable or not. If the net is controllable, Wendy generates a file "service.sa" containing the respective partner service automaton or



```
$ wendy example.net -v --sa
wendy: not using a configuration file
wendy: read net: |P|= 20  |P_in|= 5  |P_out|= 2  |T|=
wendy: message bound set to 1 (1 bytes/interface marki
wendy: send event detection requires 1 bytes per inner
wendy: writing to temporary file '/tmp/wendy-vVnmOI'
wendy: calling LoLA: 'lola-statespace /tmp/wendy-vVnmO
wendy: LoLA is done [0 sec]
wendy: closed and deleted temporary file '/tmp/wendy-v
wendy: stored 12 inner markings (2 final, 1 bad, 0 ine
wendy: stored 12 knowledges, 45 edges [0 sec]
wendy: maximal queue length: 4, final queue default le
wendy: used 12 of 65536 hash buckets, maximal bucket s
wendy: calculated 12 trivial SCCs
wendy: calculated 0 non-trivial SCCs, at most 0 member
wendy: 12 knowledges reachable
wendy: maximal simultaneously stored Knowledge objects
wendy: net is controllable: YES
wendy: writing service automaton to file 'example.sa'
wendy: closed file 'example.sa'
$ 
```

Figure 6. A screenshot of Wendy analyzing the running example.

a file "service.og" which contains the operating guideline of the service. With the command line parameter `--dot`, a graphical representation of the output is created (cf. Fig. 6). For uncontrollable nets, the diagnosis algorithm described in [15] is implemented in Wendy. This algorithm calculates an artifact similar to a witness path used in model checking tools. The discussion of the diagnosis is out of scope of this paper and the interested reader is referred to [15]. A full description of the command line parameters can be found by executing `wendy --help` or in the manual[1] which also describes the used file formats.

5.2. Integration

We have already stated in the introduction that both partner synthesis [19, 9, 3] and operating guidelines [20, 10, 13, 11, 31] can be used in many settings to solve a variety of problems in the area of service-oriented computing and business processes. To implement these advanced techniques, Wendy can be used as black-box algorithm. As such, Wendy has been integrated into several tools (see [23] for a discussion) and third party frameworks.

Most prominently, Wendy is integrated into the ProM framework [1] as a verification plugin, cf. Fig. 8(b). This makes Wendy accessible to the output of hundreds of other plugins and file formats. ProM also provides a GUI to visualize the input models and the synthesis results. Furthermore, Wendy is called by the *YAWL editor* [2] to automatically calculate correct business process configurations [3], cf. Fig 8(c). As both ProM and the YAWL editor are implemented in Java, Wendy is called as an external binary. For integration, we chose to build plugins to these tools which translate the native modeling language into a representation of our Petri net API.

To make experimental results transparent and repeatable, we further implemented a Web frontend for Wendy. It is part of the Web site service-technology.org/live [16] and is independent of a local installation

[1] Available at `http://service-technology.org/files/wendy/wendy.pdf`.

```
PLACE
  INTERNAL p0, p1, p2, p3, p4,
           p5, p6, p7, p8, p9,
           p10, omega;
  INPUT a, b, c, d, e;
  OUTPUT x, y;
  SYNCHRONOUS z;

INITIALMARKING p0;

FINALMARKING omega;

TRANSITION t1
CONSUME p0, a;
PRODUCE p1;

TRANSITION t2
CONSUME p0;
PRODUCE p2, x;

...

TRANSITION t14
CONSUME p10;
PRODUCE omega;
SYNCHRONIZE z;
```

(a) open net file format for N

```
<pnml>
  <module>
    <ports>
      <port id="port0">
        <input id="a" />
        <input id="b" />
        <input id="c" />
        <input id="d" />
        <input id="e" />
        <output id="x" />
        <output id="y" />
        <synchronous id="z" />
      </port>
    </ports>

    <net id="n1" type="PTNet">
      <place id="p0">
        <initialMarking>
          <text>1</text>
        </initialMarking>
      </place>
...
      <transition id="t1">
        <receive idref="a" />
      </transition>
...
```

(b) PNML file format for N

Figure 7. Input file formats.

Table 1. Partner synthesis for industrial WS-BPEL services.

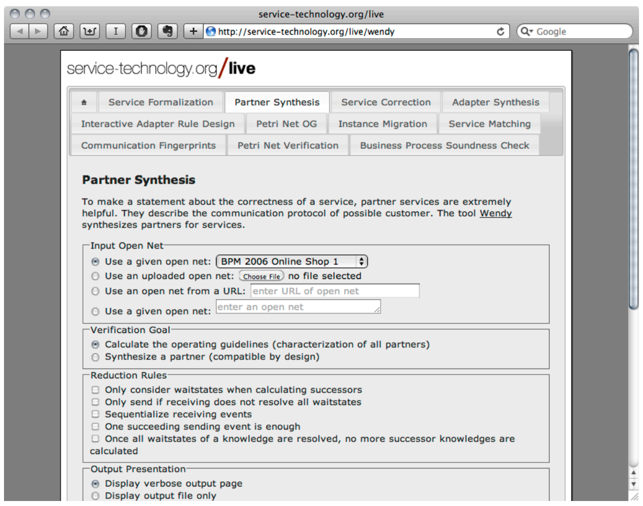
	analyzed service			synthesized partner		
service	inner markings	transitions	interface	states	transitions	time
Quotation	602	1,141	19	11,264	145,811	0
Deliver goods	4,148	13,832	14	1,376	13,838	2
SMTP protocol	8,345	34,941	12	20,818	144,940	29
Car analysis	11,381	39,865	15	1,448	13,863	49
Identity card	14,569	71,332	11	1,536	15,115	82
Product order	14,990	50,193	16	57,996	691,414	294

and provides all tools and input models used in the case study of the next section. The Wendy online demo version is accessible at `http://service-technology.org/live/wendy` where the examples of this paper can be replayed in a Web browser (cf. Fig. 8(a)). With this Web-based installation it is further possible to use Wendy as *software as a service*; that is, to call it directly over the Internet. We already used this approach to integrate LoLA [34] into the Oryx editor [7] and we plan to use the same approach to offer partner synthesis techniques to Oryx users.

5.3. Case studies

As a proof of concept, we synthesized different types of partners of several WS-BPEL services from a consulting company. Each process consists of about 40 WS-BPEL activities and models communication protocols and business processes of different industrial domains. To use Wendy, we first translated the WS-BPEL processes into open nets using the compiler BPEL2oWFN [12].

Table 1 lists details on the processes and the experimental results for the partner synthesis without applying any synthesis rules. We see that the open nets derived from the WS-BPEL processes have up to 15,000 inner markings. The interfaces consist of up to 19 message channels. The number of states of the


service-technology.org/live
http://service-technology.org/live/wendy
Google
service-technology.org/live
Service Formalization
Partner Synthesis
Service Correction
Adapter Synthesis
Interactive Adapter Rule Design
Petri Net OG
Instance Migration
Service Matching
Communication Fingerprints
Petri Net Verification
Business Process Soundness Check

Partner Synthesis

To make a statement about the correctness of a service, partner services are extremely helpful. They describe the communication protocol of possible customer. The tool Wendy synthesizes partners for services.

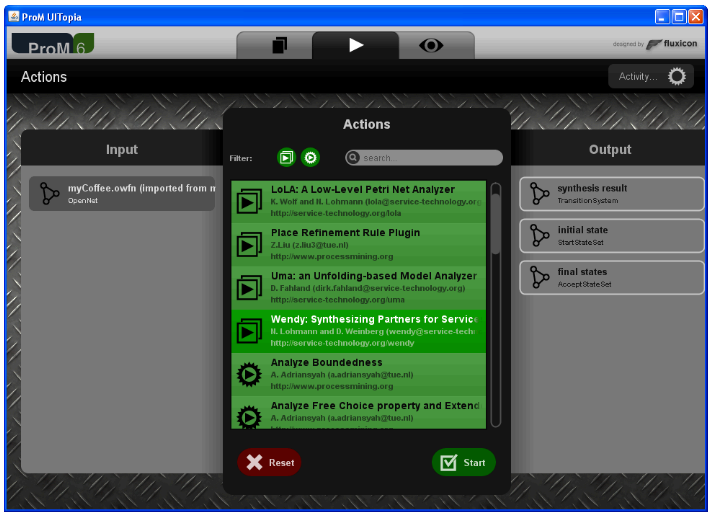
Input Open Net
- [✓ Use a given open net: BPM 2006 Online Shop 1]
- Use an uploaded open net: Choose File no file selected
- Use an open net from a URL: enter URL of open net
- Use a given open net: enter an open net

Verification Goal
- [✓ Calculate the operating guidelines (characterization of all partners)]
- Synthesize a partner (compatible by design)

Reduction Rules
- ☐ Only consider waitstates when calculating successors
- ☐ Only send if receiving does not resolve all waitstates
- ☐ Sequentialize receiving events
- ☐ One succeeding sending event is enough
- ☐ Once all waitstates of a knowledge are resolved, no more successor knowledges are calculated

Output Presentation
- [✓ Display verbose output page]
- Display output file only

(a) integration into service-technology.org/live



myCoffee.owfn (imported from m
OpenNet

LoLA: A Low-Level Petri Net Analyzer
K. Wolf and N. Lohmann (lola@service-technology.org
http://service-technology.org/lola

Place Refinement Rule Plugin
Z.Liu (z.liu3@tue.nl)
http://www.processmining.org

Uma: an Unfolding-based Model Analyzer
D. Fahland (dirk.fahland@service-technology.org)
http://service-technology.org/uma

Wendy: Synthesizing Partners for Service
N. Lohmann and D. Weinberg (wendy@service-techn
http://service-technology.org/wendy

Analyze Boundedness
A. Adriansyah (a.adriansyah@tue.nl)
http://www.processmining.org

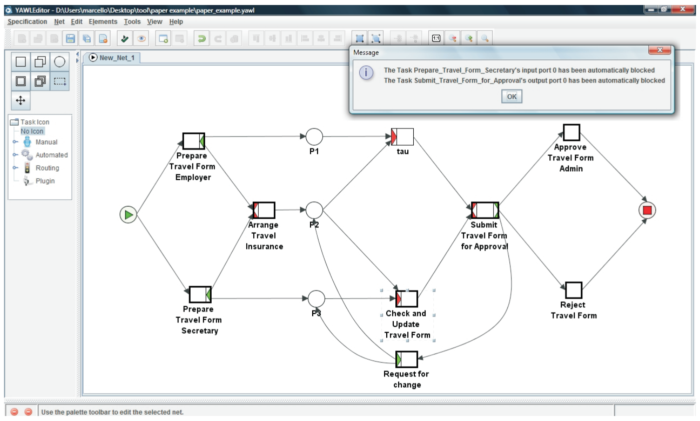
Analyze Free Choice property and Extend
A. Adriansyah (a.adriansyah@tue.nl)

synthesis result
TransitionSystem

initial state
StartStateSet

final states
AcceptStateSet

(b) integration into ProM


YAWLEditor - D:\Users\marcello\Desktop\tool\paper example\paper_example.yawl
Specification
Net
Edit
Elements
Tools
View
Help
New_Net_1
Task Icon
No Icon
Manual
Automated
Routing
Plugin
Message
OK
Prepare Travel Form Employer
P1
tau
Approve Travel Form Admin
Arrange Travel Insurance
P2
Submit Travel Form for Approval
Prepare Travel Form Secretary
P3
Check and Update Travel Form
Reject Travel Form
Request for change

The Task Prepare_Travel_Form_Secretary's input port 0 has been automatically blocked
The Task Submit_Travel_Form_for_Approval's output port 0 has been automatically blocked

Use the palette toolbar to edit the selected net.

(c) integration into the YAWL editor

Figure 8. Integration of Wendy into other tools and frameworks.

most permissive partner (or operating guideline) are sometimes much larger than the original service. The number of transitions grows even faster. The analysis takes up to 300 seconds on a 3 GHz computer with 2 GB of memory. Given that operating guidelines are usually calculated only once and are to be used by the service broker many times, this is satisfactory. Note that the calculation of the operating guideline's formulae took only a split of a second for all models.

To further evaluate Wendy, we processed parametrized academic benchmarks within a 2 GB memory limit. Figure 9 illustrates that Wendy is capable of analyzing service models with up to 5,000,000 inner markings and to synthesize partner services with up to 4,000,000 states. At the same time, we see that the largest industrial models we analyzed (14,990 inner markings and 57,996 partner states) do not come close to these bounds.

Table 2 summarizes the effect of the partner synthesis rules. Depending on the applied rule, the analysis time can be dramatically reduced. Using the rule *quit as soon as possible*, for instance, allows

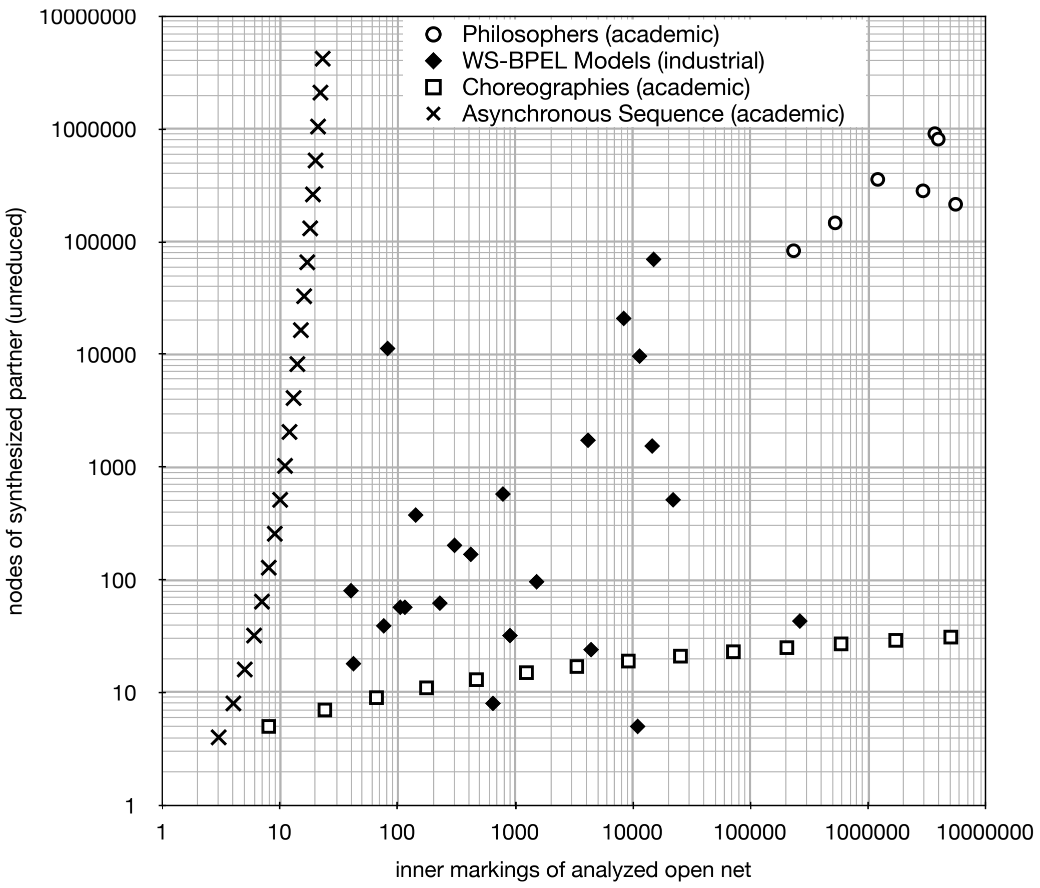


Figure 9. Limits of the synthesis algorithm (2 GB of RAM). Beside some industrial WS-BPEL models (◆), we processed several parametrized academic benchmarks: *Asynchronous sequence* (×) is a family of services with exponential growth of states of the partner services; *Choreographies* (□) are BPEL4Chor choreographies from [17] with an exponential growth of inner markings; *Philosophers* (○) is a benchmark set of the WODES workshop, see `http://www.wodes2008.org/pages/benchmark.php`.

to calculate a partner service in at most 2 seconds for all services. This result can be generalized to any other open net we analyzed so far: if the inner is bounded (controllability is undecidable if the inner is unbounded [25]), Wendy could always decide controllability by applying partner synthesis rules.

5.4. Comparison with other tools

Tools from classical controller synthesis [30] are hardly comparable to Wendy, because (1) they consider only synchronous communication (whereas Wendy also supports asynchronous communication which is crucial in SOC), (2) make different assumptions on the observability of internal states and events, and (3) do not support a concept such as an operating guideline to characterize sets of compatible partners.

Both the partner synthesis algorithm and the algorithm to calculate an operating guideline for a service have been previously implemented in the tool Fiona [26]. The design goal of Fiona was the combination of several analysis and synthesis algorithms for service behavior. This is reflected by a flexible architecture

Table 2. Partner synthesis using partner synthesis rules.

	WSO		RBS		SRE		SSE		QSP	
service	states	t	states	t	states	t	states	t	states	t
Quotation	6,244	0	1,667	0	2,287	0	52	0	20	0
Deliver goods	1,328	2	89	0	154	0	65	0	16	0
SMTP protocol	3,270	11	4,872	0	16,837	18	30	0	30	0
Car analysis	1,448	47	108	1	96	11	78	28	38	2
Identity card	1,280	74	261	1	48	2	1,280	74	12	0
Product order	57,762	300	741	1	771	3	1,782	9	101	0

which aims at the reusability of data structures and algorithms. Although this design facilitated the quick integration and validation of new algorithms, the growing complexity made optimizations more and more complicated. To overcome these efficiency problems, Wendy is a reimplementation of the synthesis algorithms as a compact single-purpose tool. This reimplementation incorporates the experiments made by analyzing performance bottlenecks through (1) improved data structures and memory management, (2) validation of case studies which gave a deeper understanding of the parameters of the models which affect scalability, and (3) theoretical observations on regularities of synthesized strategies and operating guidelines.

In comparison, Fiona could only analyze three out of the six services presented in Table 1 without taking more than 2 GB of memory. For the other services, the analysis was between 5 and 70 times slower than Wendy. In addition to Fiona, Wendy handles synchronous communication and implements two more partner synthesis rules (*succeeding sending event* and *quit as soon as possible*).

5.5. Obtain Wendy

Wendy is free software and is licensed under the GNU AGPL.[2] The source code and precompiled binaries can be downloaded from Wendy's Web site at `http://service-technology.org/wendy`. We tested several platforms including Windows, Mac OS, Linux, FreeBSD, and Solaris.

6. Conclusion

The functionality provided by Wendy — the synthesis of partners for services — is a basis of a variety of important applications in the paradigm of SOC. To this end, Wendy is already integrated into tools realizing adapter synthesis [9] and instance migration [11]. Case studies show that Wendy can be used in academic and industrial settings.

In future work, we plan to adjust the synthesis algorithm to exploit the multiple cores that are increasingly present in personal computers. In addition, symbolic representations such as BDDs [5] may further help reducing the memory consumption during the synthesis. At the same time, reduction

[2]GNU Affero Public License Version 3, `http://www.gnu.org/licenses/agpl.html`.

techniques already implemented in LoLA [34] may be applicable when calculating the inner markings of the given open net.

Acknowledgments.

The authors thank Stephan Mennicke and Christian Sura for their work on the Petri Net API and Karsten Wolf for sharing experience made with LoLA and valuable discussions on the data structures.

References

[1] Aalst, W. M. P. v. d., Dongen, B. F. v., Günther, C. W., Mans, R. S., de Medeiros, A. K. A., Rozinat, A., Rubin, V., Song, M., Verbeek, H. M. W. E., Weijters, A. J. M. M.: ProM 4.0: Comprehensive Support for *real* Process Analysis, *ICATPN 2007*, LNCS 4546, Springer, 2007.

[2] Aalst, W. M. P. v. d., Hofstede, A. H. M. t.: YAWL: yet another workflow language, *Inf. Syst.*, **30**(4), 2005, 245–275.

[3] Aalst, W. M. P. v. d., Lohmann, N., La Rosa, M., Xu, J.: Correctness Ensuring Process Configuration: An Approach Based on Partner Synthesis, *BPM 2010*, LNCS 6336, Springer-Verlag, 2010.

[4] Alonso, G., Casati, F., Kuno, H., Machiraju, V.: *Web Services: Concepts, Architectures and Applications*, Springer, 2003.

[5] Bryant, R. E.: Graph-Based Algorithms for Boolean Function Manipulation, *IEEE Trans. Computers*, **C-35**(8), 1986, 677–691.

[6] Cortadella, J., Kishinevsky, M., Kondratyev, A., Lavagno, L., Yakovlev, A.: Petrify: A Tool for Manipulating Concurrent Specifications and Synthesis of Asynchronous Controllers, *Trans. Inf. and Syst.*, **E80-D**(3), March 1997, 315–325.

[7] Decker, G., Overdick, H., Weske, M.: Oryx - An Open Modeling Platform for the BPM Community, *BPM 2008*, LNCS 5240, Springer, 2008.

[8] DeRemer, F., Kron, H. H.: Programming-in-the-Large Versus Programming-in-the-Small, *IEEE Trans. Software Eng.*, **2**(2), 1976, 80–86.

[9] Gierds, C., Mooij, A. J., Wolf, K.: *Specifying and generating behavioral service adapter based on transformation rules*, Preprint CS-02-08, Universität Rostock, Rostock, Germany, 2008.

[10] Kaschner, K., Lohmann, N.: Automatic Test Case Generation for Interacting Services, *ICSOC 2008 Workshops*, LNCS 5472, Springer, 2009.

[11] Liske, N., Lohmann, N., Stahl, C., Wolf, K.: Another Approach to Service Instance Migration, *ICSOC 2009*, LNCS 5900, Springer, 2009.

[12] Lohmann, N.: *A Feature-Complete Petri Net Semantics for WS-BPEL 2.0 and its Compiler BPEL2oWFN*, Informatik-Berichte 212, Humboldt-Universität zu Berlin, Berlin, Germany, August 2007.

[13] Lohmann, N.: Correcting Deadlocking Service Choreographies Using a Simulation-Based Graph Edit Distance, *BPM 2008*, LNCS 5240, Springer-Verlag, 2008.

[14] Lohmann, N.: A Feature-Complete Petri Net Semantics for WS-BPEL 2.0, *WS-FM 2007*, LNCS 4937, Springer, 2008.

[15] Lohmann, N.: Why does my service have no partners?, *WS-FM 2008*, LNCS 5387, Springer-Verlag, 2009.

[16] Lohmann, N.: service-technology.org/live – Replaying tool experiments in a Web browser, *BPM 2010 Demos*, 615, CEUR-WS.org, 2010.

[17] Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and Participant Synthesis, *WS-FM 2007*, LNCS 4937, Springer, 2008.

[18] Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing Interacting WS-BPEL Processes Using Flexible Model Generation, *Data Knowl. Eng.*, **64**(1), 2008, 38–54.

[19] Lohmann, N., Massuthe, P., Wolf, K.: Behavioral Constraints for Services, *BPM 2007*, LNCS 4714, Springer-Verlag, 2007.

[20] Lohmann, N., Massuthe, P., Wolf, K.: Operating Guidelines for Finite-State Services, *PETRI NETS 2007*, LNCS 4546, Springer, 2007.

[21] Lohmann, N., Mennicke, S., Sura, C.: The Petri Net API: A collection of Petri net-related functions, *AWPN 2010*, 643, CEUR-WS.org, 2010.

[22] Lohmann, N., Weinberg, D.: Wendy: A tool to synthesize partners for services, *31st International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, PETRI NETS 2010, Braga, Portugal, June 21–25, 2010, Proceedings* (J. Lilius, W. Penczek, Eds.), 6128, Springer-Verlag, June 2010.

[23] Lohmann, N., Wolf, K.: How to Implement a Theory of Correctness in the Area of Business Processes and Services, *BPM 2010*, LNCS 6336, Springer-Verlag, 2010.

[24] Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA, *Annals of Mathematics, Computing & Teleinformatics*, **1**(3), 2005, 35–43.

[25] Massuthe, P., Serebrenik, A., Sidorova, N., Wolf, K.: Can I find a Partner? Undecidablity of Partner Existence for Open Nets, *Inf. Process. Lett.*, **108**(6), 2008, 374–378.

[26] Massuthe, P., Weinberg, D.: FIONA: A Tool to Analyze Interacting Open Nets, *AWPN 2008*, CEUR Workshop Proceedings Vol. 380, CEUR-WS.org, 2008.

[27] Mcilroy, D.: Mass-produced Software Components, *Software Engineering Concepts and Techniques*, NATO Science Committee, 1969.

[28] Papazoglou, M. P.: Agent-oriented technology in support of e-business, *Commun. ACM*, **44**(4), 2001, 71–77.

[29] Papazoglou, M. P.: *Web Services: Principles and Technology*, Pearson - Prentice Hall, 2007.

[30] Ramadge, P., Wonham, W.: The control of discrete-event systems, *Proceedings of the IEEE*, **77**(1), 1989, 81–98.

[31] Stahl, C., Massuthe, P., Bretschneider, J.: Deciding Substitutability of Services with Operating Guidelines, *LNCS ToPNoC*, **II**(5460), 2009, 172–191.

[32] Szyperski, C.: *Component Software–Beyond Object-Oriented Programming*, Addison-Wesley and ACM Press, 1998.

[33] Weinberg, D.: Efficient Controllability Analysis of Open Nets, *WS-FM 2008*, LNCS 5387, Springer, 2009.

[34] Wolf, K.: Generating Petri Net State Spaces, *PETRI NETS 2007*, LNCS 4546, Springer, 2007, Invited lecture.

[35] Wolf, K.: Does my service have partners?, *LNCS ToPNoC*, **5460**(II), 2009, 152–171.