# Analysis Techniques for Service Models

*(Invited Paper)*

Wolfgang Reisig, Dirk Fahland, Niels Lohmann, Peter Massuthe, Christian Stahl, Daniela Weinberg
Humboldt-Universität zu Berlin
Institut für Informatik
{reisig, fahland, nlohmann, massuthe, stahl, weinberg}@informatik.hu-berlin.de
Karsten Wolf, Kathrin Kaschner
Universität Rostock
Institut für Informatik
{karsten.wolf, kathrin.kaschner}@uni-rostock.de

***Abstract*—The paradigm of *Service-Oriented Computing* (SOC) provides a framework for interorganizational business processes and for the emerging *programming-in-the-large*. The basic idea of SOC, the *interaction* of services, rises a lot of issues such as proper termination of interacting services or substitution of a service by another one. Such issues can be addressed by means of *models* of services. We show how services can intelligibly be modeled, and we present algorithms and tools to analyze properties of service models. In order to emphasize that our models properly reflect real world issues of services, we also show that services represented in established languages such as WS-BPEL can be transformed into our formal method.**

***Index Terms*—services, service-oriented architectures, service modeling, formal analysis**

## I. Introduction

The paradigm of *Service-Oriented Computing* (SOC) [19] provides a framework for interorganizational business processes and for the emerging *programming-in-the-large* [6]. The basic notion of SOC is the *interaction* of services: Services can interact, resulting in a new service. Service-Oriented Architectures (SOA) [7] provide a means for organizing service interaction. This service interaction can be depicted as a triangle (see Fig. 1). Here, we assume *requesters, providers*, and *brokers* of services. A service requester runs services that do not properly function in isolation, but rather require interaction (*binding*) with other services. A service provider may offer such a service. The problem is to identify requester and provider services that would properly bind. This is what a broker does: The requester informs the broker about the service he wants to *find*. The provider *publishes* descriptions (*public views* [2], [11] or *operating guidelines* [16], [17], [18]) of the services he offers to the broker as well. The broker then matches the service descriptions and informs both, the requester and the provider, about properly binding pairs of their services.

Those activities require clear-cut notions of services, proper binding, public views etc, as well as efficient algorithms to decide, based on abstract descriptions, which services would properly bind. More specifically formulated, a number of problems arises, including the following ones:

- Are two services partners, i.e. do they compose properly?
- Does a service have partners at all (otherwise, it is definitely ill-designed)?
- How to characterize all partners of a service?
- How to represent an operating guideline for a service?
- Is there a canonical (most liberal) partner of a service?
- Can a service $S'$ substitute a service $S$?
- Is there a canonical (most abstract) $S'$ to substitute $S$?
- Can $S'$ substitute $S$ at runtime?
- How to adapt (i.e., mediate between) two services that "almost" properly interact?

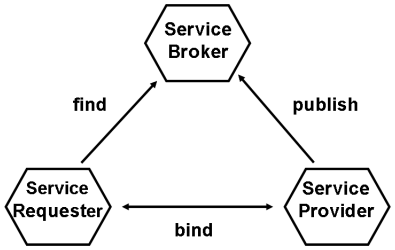


Fig. 1. The SOA triangle.

Such questions can be addressed by means of *models* of services. Models allow to abstract from unnecessary details, highlighting the notions and constructs that are relevant in the given context.

This paper surveys ongoing research on the issues described above, as conducted at Humboldt-Universität zu Berlin and Universität Rostock. The long-term aims of this research comprises clear-cut models of services, their interaction, abstract description, the characterization of *sets* of services, as well as efficient algorithms to decide questions on services.

## II. Services

Service-Oriented Computing combines existing, well established technologies, in particular communication protocols and middleware. Languages such as WS-BPEL [3] support the description and execution of services.

It has turned out useful to distinguish three aspects of services on an abstract level: A service has

- an *id* that makes it unique,
- an interface to its environment, and

- an internal behavior.

In the special case of a *web service*, the *id* is an URI and the interface is specified in the Web Service Description Language, WSDL [5].

## III. Service Interaction

The purpose of a service $S$ is to offer a distinguished functionality to its environment. A service $R$ in the environment of $S$ can solicit this functionality in the course of interacting with $S$. Interaction of $S$ and $R$ may *succeed* or *fail*. Matching of services can be defined in different ways:

- Their *interfaces* match: This just means, that their interfaces consist of typed *input* and *output ports* such that each output port of one service corresponds to exactly one likewise typed input port of the other service.
- Their *behaviors* match: This additionally states that each interaction makes sense. So, each service has a proper reaction to each message received. Technically speaking, the interaction terminates properly. That means no livelock occurs and both services reach a proper final state.
- Their *semantics* match: In the course of an interaction it may be that a service can not properly interpret a received message and needs additional information to adapt the message's contents to its own standards. This is what the *semantic web* approach [4], [9], [23] supports.
- Their *quality of service* matches: In addition to the requirements above, nonfunctional properties can be specified. For example, a service may expect arriving messages to meet a timeline or may require the partner service to be fault tolerant. Further examples include requirements such as transactional behavior and compensation, as it is well known for data bases.

This paper focuses on the second aspect, i.e., matching behavior.

## IV. Service Modeling

There are many reasons to *model* services, independent of implementation details. First of all, a service should be comprehensible without reference to its implementation. This is most obvious for services which are not intended to be implemented. The SOC paradigm supports this view. Furthermore, a service should be conceivable by human. Hence, it should be represented in a form readable by human.

But models are useful for implemented services, too. The situation resembles programming languages, where the semantics of a program should not depend on a specific compiler.

Finally, one expects a service to fulfill specific requirements, i.e., to have specific properties. It should be possible to formulate such properties correctly and to prove that a service indeed meets them. All these demands can be achieved with the help of proper modeling and analysis techniques with software tools that support handling and analysis of models. We present such techniques in the sequel.

As a modeling technique, we suggest *open workflow nets* (oWfN) [16], a special class of Petri nets. oWfN equip van der Aalst's workflow nets [1] with an *interface*, i.e., a set of input and output places. Interaction of two services is modeled as *composition* of their oWfN models. To compose two oWfN $R$ and $S$, some of the two nets' interface places are identified. The resulting net $R \oplus S$ is an oWfN, again. The quest of proper binding of two services can be seen as the quest of whether the composed net $R \oplus S$, gained from the oWfN models $R$ and $S$ of the two given services, will properly terminate. To name this issue, $R$ and $S$ are *partners* iff $R \oplus S$ properly terminates.

We refrain from formal details here; the forthcoming examples show the principles.

## V. Toy Example: A Model of a Vending Machine

The forthcoming toy example is intended to reveal a lot of interesting and partly fairly subtle questions occurring even on the simple level of service composition and proper termination.

We consider a simple version of a *vending machine*. The machine has a slot for coins and two buttons, called coffee! and tea!. When a coin has been inserted and one of the buttons has been pressed, the machine delivers the corresponding beverage and reaches a final state.

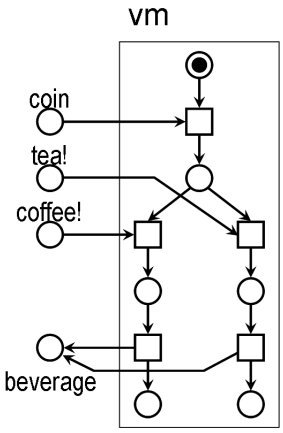


Fig. 2. The model vm of a vending machine.

Figure 2 depicts an oWfN model, vm, of this vending machine. As a convention for oWfN models, the behavior is represented inside the box and the interface outside the box. Apparently no transition of vm is enabled in the given initial state. Intuitively formulated, the machine waits for a coin to be inserted.

Figure 3 shows a model of a "coffee partner", cp, soliciting coffee from vm: He or she inserts a coin, presses the coffee! button, and then waits for the coffee.

Composition of oWfN is simply modeled as merging equally named interface places. Consequently, Fig. 4 depicts the composition cp $\oplus$ vm of the two service models. cp $\oplus$ vm properly terminates with a token on the final places of both service components.

According to the above definition, vm and cp are partners. We will now investigate the set *partners*(vm) of *all* partners of the vending machine. The structure of the "tea partner" tp in Fig. 5 should not be surprising.

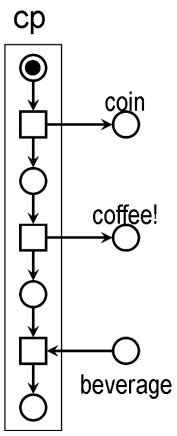


Fig. 3. The model cp of the coffee partner.

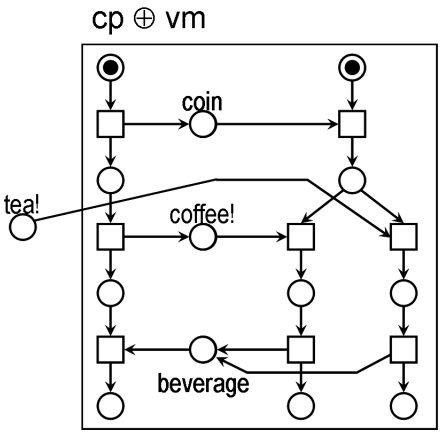


Fig. 4. The composition cp ⊕ vm.

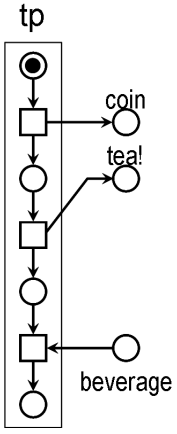


Fig. 5. The model tp of the tea partner.

Furthermore, a partner may decide on the beverage *after* inserting the coin: the "coffee-or-tea-partner" ctp in Fig. 6.

Does vm have any further partners? There are more partners indeed – based on the observation that the user may very well push a button *before* inserting the coin. This yields three other partners. Fig. 7 shows one of them, cp'. The corresponding

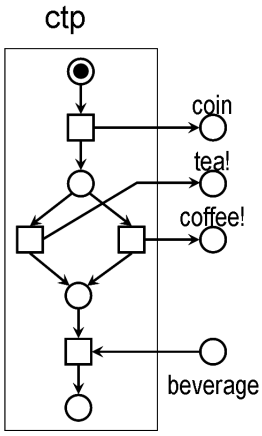


Fig. 6. The model ctp of the coffee-and-tea-partner.

partners tp' and ctp' are then obvious.

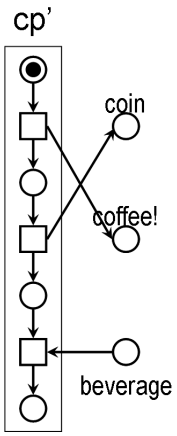


Fig. 7. The model cp' of the variant of the coffee partner.

We are still missing another partner. In fact, we look for the most important one – the canonical partner. Its behavior is based on the observation that inserting a coin is detached from pushing a button: No order is required at all, both actions can occur concurrently, in different threads of control. And accepting the beverage is organized in a further thread of control.

Figure 8 shows this partner. It is the most *distributed partner*, mdp, of the vending machine: Every other partner orders the transitions more strictly.

We have not yet reached the end of the story. In general, a service is not linked to only *one* other service, but to several others. To model this aspect, we partition the service's interface into disjoint subsets, called *ports*. Each port can be attached to a different service. For example, Fig. 9 partitions the vending machine's interface into three ports.

Figure 10 shows a typical example of three services, called dad, mom, and kid, for obvious reasons. Together with vm, there now are four services that jointly terminate properly.

The dad, mom, and kid services act *autonomously*. This is a subtle aspect, best conceivable by a counterexample. To this

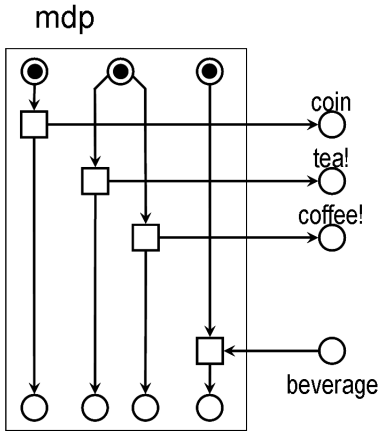


Fig. 8. The model mdp of the most distributed partner.

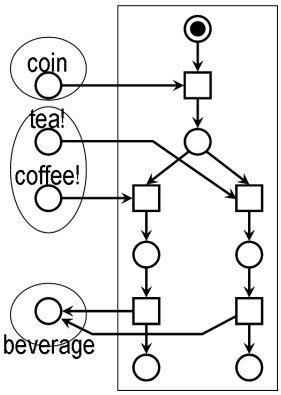


Fig. 9. The vending machine with ports.

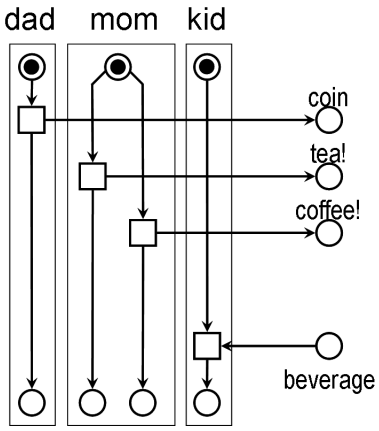


Fig. 10. The dad, mom and kid services.

end, we refine the tea! port and the coffee! port of Fig. 9, as shown in Fig. 11.

Fig. 12 represents four partners of this service, namely, one to insert the coin, one to press the tea! button (this one does nothing), one to press the coffee! button, and one to receive the beverage. Together with these partners the service of Fig. 11 terminates properly. However, if the second partner would

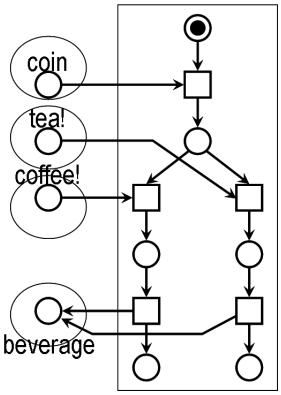


Fig. 11. Re-shuffled ports.

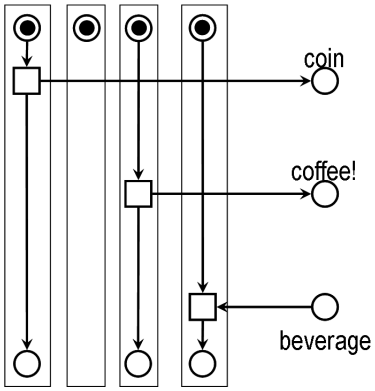


Fig. 12. Four partners for Fig. 11.

indeed be able to press the tea! button, he would have to *coordinate* his action with the third partner: Exactly one of them must press his or her button. So, the service of Fig. 12 has no autonomously acting partners.

## VI. Questions on the Partners of a Service

The above example revealed subtle properties of services. They are to be analyzed in models, far before implementation.

For a given oWfN $S$, a number of practically important questions on the set $partners(S)$ of all partners of $S$ can be formulated and analyzed with the help of oWfN:

1) *Composability*: Is a (further) given oWfN, $R$, a partner of $S$? This can be figured out by analyzing the (finite) state space of $R \oplus S$.
2) *Controllability*: Does $S$ have partners at all? Figure 13 shows an oWfN without any partner: A partner would have to correctly "guess" the outcome of an internal choice of $S$. Algorithms to decide controllability have been published in [15], [21], [24].
3) *Canonical partner*: It would be quite helpful if among all partners of $S$ there was one with particularly useful properties. A most distributed partner would be a good candidate. Algorithms to construct this partner have been defined. Details can be found in [21].

4) *Operating Guideline*: A potential partner of $S$ must know what he or she may do to act as a partner of $S$. This is usually described in the *operating guideline* of $S$. Technically, this is achieved by the *most distributed* partner, equipped with some additional information. For details please refer to [16], [17], [18].

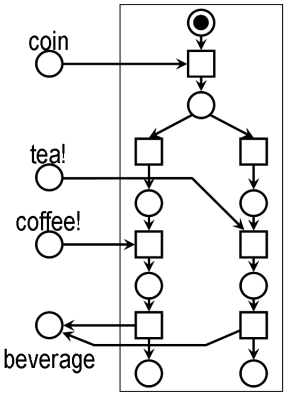


Fig. 13. A service without partners.

In [20] we propose a tool chain and algorithms to decide some of the questions stated above. In [12] we present the tool Fiona, which decides controllability and computes the operating guideline. Since we want to decide important questions about a given service efficiently, we have enriched Fiona with several optimizations. To decide controllability, Fiona applies a number of reduction rules as a partner of the given oWfN is computed [24]. The operating guideline can be constructed symbolically using BDDs (Binary Decision Diagrams) [10], which is more efficient and needs less resources.

## VII. Questions at the Substitution of a Service

The provider of a service $S$ may have good reasons to replace $S$ by a service $S'$: $S'$ may perform in a better way, offer more proper behavior, run on new hardware, or integrate more conveniently into a new architecture, etc. Nevertheless, the current partners of $S$ should not be bothered, they should not even be aware of the substitution. So, it is reasonable to define for services $S$ and $S'$: $S$ can be substituted by $S'$ iff *partners*$(S) \subseteq$ *partners*$(S')$.

Again, a bunch of questions on substitution can be formulated and analyzed with the help of oWfN:

1) *Substitution*: Can a given service $S'$ substitute $S$? This is not a trivial problem, as the sets $partners(S)$ and $partners(S')$ are in general infinite.
2) *Migration*: Can $S'$ substitute $S$ at runtime? This is important for long running services. An example is the service of a life insurance contract. Particular measures are needed to continue with $S'$ as $S$ was running before.
3) *Canonical Substitute*: Is there a canonical service $S'$ to substitute $S$? A reasonable candidate for $S'$ is a public view of $S$, hiding all irrelevant details, but behaving like $S$. Unfortunately, there exists no unique public view in general [14].

## VIII. Further Questions on Services

A lot of further questions on services can properly be formulated and analyzed with help of models, in particular with help of oWfN. Here, we identify five of them:

1) *Adapter generation*: As already mentioned in Sect. III, two services may not be partners in the strict sense. For instance, they may require adaption of mutual messages. An *adapter* service may help. Technically this means for two given oWfN $R$ and $S$, we need an oWfN $A$, such that $R \oplus A \oplus S$ properly terminates.
2) *ACID properties*: The principle of "all or nothing" and the necessity to occasionally undo what has already been done, are well known from data bases. They are likewise demanded in the context of services. There, they are particularly difficult to be realized if they span more than one service. So, the properties have to be realized by passing messages between the involved services.
3) *Outsourcing*: Service-Oriented Computing supports the flexibility of composition and substitution. Hence, the idea to *outsource* some components of a service is natural. Theory supported techniques to correctly outsource components of a service are not available, yet.
4) *Compliance*: Services in the business world have to comply to a lot of rules given from outside. Verification of compliance remains an open problem.

## IX. Models of WS-BPEL Processes

So far we have shown our formal model of oWfN and introduced analysis questions on that formal model. In the following, we show how the concepts can be applied to real world services described with industrial languages. As an example, we use the business process description language WS-BPEL [3], but any other language could be used as well.

WS-BPEL supports the orchestration of web services. A WS-BPEL process describes the control flow of a business process, in particular receiving and sending of messages from and to the process environment. In addition, WS-BPEL provides means to handle faults and to compensate already executed actions that must be made undone.

The oWfN formalism is powerful enough to formulate a feature complete semantics of WS-BPEL, as we did in [8], [22]. In particular this includes fault handling and compensation handling. As an example, the XML code snippet of Fig. 14 depicts a WS-BPEL receive activity. Figure 15 shows the corresponding oWfN. It also includes stop and stopped interface places, thus providing means for the handling of faults and compensation.

The translation of WS-BPEL processes into oWfN essentially follows the block structure of WS-BPEL. Details are given in [22].

Choice and order of WS-BPEL activities frequently depend on the content of messages. This requires oWfN with colored tokens. We refrain from details here.

```
<receive
  partnerLink="customer"
  portType="customerPT"
  operation="login"
  variable="var">
</receive>
```

Fig. 14. A `receive` activity.

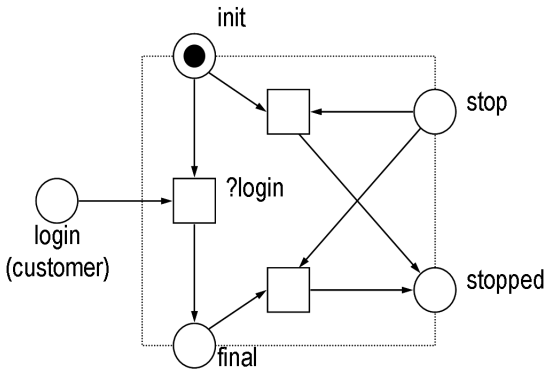


Fig. 15. oWfN representation of Fig. 14.

We have developed and implemented the tool BPEL2oWFN, that automatically transforms a WS-BPEL process into an oWfN. That way we are able to analyse any given WS-BPEL process using BPEL2oWFN and Fiona as proposed in [12], [13].

## X. Conclusion

Service-Oriented Computing requires solid foundations based on clear definitions of basic notions, and of fundamental properties. In particular, the requirements of the SOA triangle must be made precise and operational. This rises a lot of problems on partners of a service and of the service's substitutes. Here, efficient algorithmic solutions are mandatory for the industrial design of Service-Oriented Architectures.

This paper compiled and exemplified some founding notions and properties of services, as suggested by the research groups at Humboldt-Universität zu Berlin and at Unversität Rostock. It has been shown that open Workflow Nets provide a solid basis for a lot of algorithms to answer significant questions at the build-time of a service. The usefulness of the approach in practice has been demonstrated by a number of case studies and by the feature complete translation of WS-BPEL processes into oWfN.

## References


[1] Wil van der Aalst. The application of Petri nets to workflow management. *Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[2] Wil van der Aalst and Mathias Weske. The P2P approach to interorganizational workflows. In *Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01)*, volume 2068 of *Lecture Notes in Computer Science*, pages 140–156. Springer-Verlag, 2001.

[3] Alexandre Alves, Assaf Arkin, Sid Askary, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guízar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter König, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, and Alex Yiu. Web Services Business Process Execution Language Version 2.0. Public Review Draft 02, 20 November, 2006, OASIS, November 2006.

[4] Saartje Brockmans, Marc Ehrig, Agnes Koschmider, Andreas Oberweis, and Rudi Studer. Semantic Alignment of Business Processes. In Y. Manolopoulos; J. Filipe; P. Constantopoulos; J. Cordeiro, editor, *Proceedings of the Eighth International Conference on Enterprise Information Systems (ICEIS 2006)*, pages 191–196, Paphos, Cyprus, MAY 2006. INSTICC Press.

[5] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Service Discription Language (WSDL) 1.1. Technical report, Ariba, International Business Machines Corporation, Microsoft, March 2001.

[6] Frank DeRemer and Hans H. Kron. Programming-in-the-large versus programming-in-the-small. *IEEE Trans. Software Eng.*, 2(2):80–86, 1976.

[7] Karl Gottschalk. Web Services architecture overview. IBM whitepaper, IBM developerWorks, September 2000. http://ibm.com/developerWorks/web/library/w-ovr/.

[8] Sebastian Hinz, Karsten Schmidt, and Christian Stahl. Transforming BPEL to Petri nets. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proceedings of the Third International Conference on Business Process Management (BPM 2005)*, volume 3649 of *Lecture Notes in Computer Science*, pages 220–235, Nancy, France, September 2005. Springer-Verlag.

[9] Yannis Kalfoglou and W. Marco Schorlemmer. Ontology Mapping: The State of the Art. In Yannis Kalfoglou, W. Marco Schorlemmer, Amit P. Sheth, Steffen Staab, and Michael Uschold, editors, *Semantic Interoperability and Integration*, volume 04391 of *Dagstuhl Seminar Proceedings*. IBFI, Schloss Dagstuhl, Germany, 2005.

[10] Kathrin Kaschner, Peter Massuthe, and Karsten Wolf. Symbolische Repräsentation von Bedienungsanleitungen für Services. In Daniel Moldt, editor, *13. Workshop "Algorithmen und Werkzeuge für Petrinetze" (AWPN 2006), Proceedings*, pages 54–61. Universität Hamburg, September 2006. in German.

[11] Frank Leymann, Dieter Roller, and Marc-Thomas Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2):198–211, 2002.

[12] Niels Lohmann, Peter Massuthe, Christian Stahl, and Daniela Weinberg. Analyzing Interacting BPEL Processes. In *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, volume 4102 of *Lecture Notes in Computer Science*, pages 17–32. Springer-Verlag, September 2006.

[13] Niels Lohmann, Peter Massuthe, Christian Stahl, and Daniela Weinberg. Analyzing Interacting WS-BPEL Processes. *Data & Knowledge Engineering (DKE)*, 2007. to appear.

[14] Axel Martens. Algorithmic Generation of the Public View. Technical report, Humboldt-Universität zu Berlin, March 2002.

[15] Axel Martens. Analyzing Web Service based Business Processes. In Maura Cerioli, editor, *Proceedings of Intl. Conference on Fundamental Approaches to Software Engineering (FASE'05), Part of the 2005 European Joint Conferences on Theory and Practice of Software (ETAPS'05)*, volume 3442 of *Lecture Notes in Computer Science*, pages 19–33, Edinburgh, Scotland, April 2005. Springer-Verlag.

[16] Peter Massuthe, Wolfgang Reisig, and Karsten Schmidt. An Operating Guideline Approach to the SOA. *Annals of Mathematics, Computing & Teleinformatics*, 1(3):35–43, 2005.

[17] Peter Massuthe and Karsten Schmidt. Operating Guidelines – An Automata-Theoretic Foundation for the Service-Oriented Architecture. In Kai-Yuan Cai, Atsushi Ohnishi, and M.F. Lau, editors, *Proceedings of the Fifth International Conference on Quality Software (QSIC 2005)*, pages 452–457, Melbourne, Australia, September 2005. IEEE Computer Society.

[18] Peter Massuthe and Karsten Wolf. An Algorithm for Matching Nondeterministic Services with Operating Guidelines. In Frank Leymann, Wolfgang Reisig, Satish R. Thatte, and Wil M. P. van der Aalst, editors, *The Role of Business Processes in Service Oriented Architectures*, number 06291 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[19] Mike P. Papazoglou. Agent-oriented technology in support of e-business. *Communications of the ACM*, 44(4):71–77, 2001.

[20] Wolfgang Reisig, Karsten Schmidt, and Christian Stahl. Kommunizierende Workflow-Services modellieren und analysieren. *Informatik - Forschung und Entwicklung*, pages 90–101, October 2005.

[21] Karsten Schmidt. Controllability of Open Workflow Nets. In Jörg Desel and Ulrich Frank, editors, *Enterprise Modelling and Information Systems Architectures*, number P-75 in Lecture Notes in Informatics (LNI), pages 236–249. Entwicklungsmethoden für Informationssysteme und deren Anwendung (EMISA, RWTH Aachen), Bonner Köllen Verlag, 2005.

[22] Christian Stahl. A Petri Net Semantics for BPEL. Techn. Report 188, Humboldt-Universität zu Berlin, July 2005.

[23] The OWL Services Coalition. OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/, 2003.

[24] Daniela Weinberg. Reduction Rules for Interaction Graphs. Techn. Report 198, Humboldt-Universität zu Berlin, February 2006.