Service Discovery Using Communication Fingerprints*

Olivia Oanea[1], Jan Sürmeli[2], and Karsten Wolf[1]

[1] Universität Rostock
18051 Rostock, Germany
{olivia.oanea,karsten.wolf}@uni-rostock.de
[2] Humboldt-Universität zu Berlin
Unter den Linden 6
10099 Berlin, Germany
suermeli@informatik.hu-berlin.de

Abstract. A request to a service registry must be answered with a service that fits in several regards, including semantic compatibility, non-functional compatibility, and interface compatibility. In the case of stateful services, there is the additional need to check behavioral (i.e. protocol) compatibility. This paper is concerned with the latter aspect. For speeding up compatibility checks which need to be performed on many candidate services, we propose an abstraction of the behavior of each published service that we call communication fingerprint. The technique is based on linear programming and is thus extremely efficient. We validate our approach on a large set of services that we cut out of real world business processes.

1 Introduction

In a service oriented architecture, we expect a service *broker* to manage a service *registry*. The broker can be approached by a service *provider* or a service *requester*. Service providers want their service to be published such that it can later on be bound to a service requester. The service broker may extract useful information about the provided service. A service requester approaches the broker for extracting one of the registered services. Besides all kind of functional and non-functional properties that should match the request, it is important that the requesting service R and the service P selected by the broker have compatible behavior, i.e. their interaction should not run into problems such as deadlocks and livelocks. In this article, we propose an approach for supporting a service broker in this regard.

An apparent method for asserting deadlock and livelock freedom would be to model check [2] the composition $R \oplus P$ before shipping the URI of P to R. This is a rather expensive procedure which has a strong negative impact on the response time of the broker. For this purpose, we proposed an alternative check [3] which preprocesses fragments of the state space to be checked at publish time. However, even this check must in worst case be applied to all compositions $R \oplus P_i$ where $\{P_1, \ldots, P_n\}$ is the set of registered services. Consequently, we need a complementing technique which narrows the set of registered services to be checked with R to a subset as small as possible.

* An extended version of this paper is available as technical report [1].

P.P. Maglio et al. (Eds.): ICSOC 2010, LNCS 6470, pp. 612–618, 2010.

To this end, we propose *communication fingerprints*. A communication fingerprint of service P collects constraints on the number of occurrences of messages in any correct run of a composed system that involves P. The fingerprint of a registered service is computed upon publishing a service. When a requester approaches the registry, its fingerprint is computed as well and matched with the fingerprints of the registered services. We show that fingerprint matching is a necessary condition for correct interaction, so the expensive model checking procedures need only be executed for matching services.

For computing communication fingerprints, we rely on Petri net models of services which can be automatically obtained [4] from specifications in the industrial language WS_BPEL [5]. We apply a technique called the *state equation* to the Petri net models. The state equation provides a linear algebraic relation between two markings (states) m and m' as well as a sequence of transitions that transforms m into m'. Using the state equation, we derive constraints on the number of message occurrences. For matching fingerprints, we rely on a relation between the state equations of components and the state equation of the composed system that has been observed in [6,7,8].

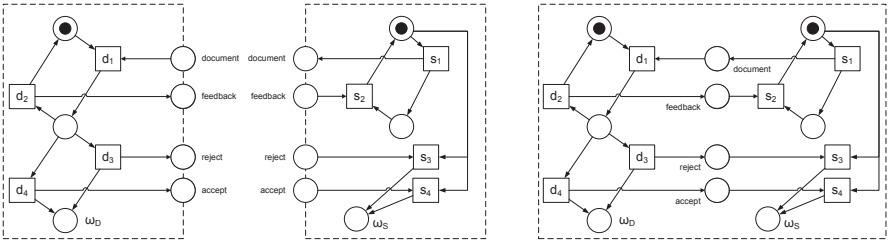
The paper is organized as follows. We introduce Petri net models for services and formally define compatibility of services. We continue with the formal definition of fingerprints, their computation, and their application in deciding compatibility. We present a case study that underpins the performance gain of our approach. Finally, we discuss further use cases for communication fingerprints.

2 Compatibility of Services

We model services as *open nets*, which are Petri net models with classical syntax and semantics, enriched with a simple notion to support message exchange and composition. Figure 1 shows two example open nets D and S and their composite $\mathsf{D} \oplus \mathsf{S}$: Initially, transition s_1 is enabled. Firing s_1, a message is sent by S over channel document, enabling transition d_1. Firing d_1, D consumes the message from the document channel and results in a state that allows D to make a decision: It fires either d_2, d_3 or d_4, resulting in a message either on the channel feedback, reject or accept. If d_2 fires, D is in a state waiting for a new message over channel document; otherwise it is in a final state $[\omega_{\mathsf{D}}]$. Before this decision, S is in a state waiting for a message over channel feedback.

The composite $\mathsf{S} \oplus \mathsf{D}$ can not reach a common final state from any other reachable state. We thus call D and S *incompatible*: In this paper, we inspect the compatibility criterion *weak termination* of a closed composite. Weak termination is similar to *soundness* which is applied as a correctness criterion in the field of business processes. A closed composite is *weakly terminating*, if from each reachable state, some final state is *reachable*. We call two partner services N and Q *compatible* if either (1) their composite is closed and weakly terminating or (2) there exists a partner P of $N \oplus Q$, such that $(N \oplus Q) \oplus P$ is closed and weakly terminating. For finite state services, (1) can be decided by model checking; case (2) is exploited in [9].

Since all known techniques to decide compatibility base on state space computation, they result in a high complexity and runtime. In our scenario, many such decisions need to be done subsequently. We thus suggest a necessary condition for compatibility to preselect potential partners. The condition is based on the observation that, in a terminating



(a) D **(b)** S **(c)** D ⊕ S

Fig. 1. Two example services D and S, and their composition D⊕S. The final markings are defined as follows: $\Omega_{\mathsf{S}} = \{[\omega_{\mathsf{S}}]\}$, $\Omega_{\mathsf{D}} = \{[\omega_{\mathsf{D}}]\}$, and $\Omega_{\mathsf{D}\oplus\mathsf{S}} = \{[\omega_{\mathsf{D}}, \omega_{\mathsf{S}}]\}$.

run, the number of *send* events for some message type must match the number of *receive* events for that type. Consequently, it is necessary for compatibility that the range of send event occurrences permitted by the internal control flow of one service must intersect with the range of receive event occurrences permitted by the internal control flow of the other service. The same is true for more complicated expressions involving send and receive events, e.g. the difference between the number of send events. Such more complicated expressions are quite useful for services with loops where the absolute number of occurrences of some event may be unbounded. The mentioned ranges for occurrence numbers can be computed for each service independently.

3 Communication Fingerprints

The objective of a fingerprint [1] is to represent the terminating behavior of an open net finitely on an abstraction level which is applicable for preselecting a set of compatible partner services. Since compatibility bases on proper interaction, we focus on *message dependencies* and abstract from private activites. More precisely, we take into account boolean combinations of integer linear constraints, counting messages that are sent over specific channels and the relations thereof.

Formally, a *linear constraint* $c = f * b$ consists of a formal sum f containing channel name variables, a comparison operator $* \in \{\leq, \geq, =, <, >\}$, and an integer b. To decide whether a transition sequence σ *satisfies* a linear constraint c, we abstract σ to its channel usage $\alpha(\sigma)$: A function mapping each channel a to the number of messages exchanged over a while executing σ. Obviously, $\alpha(\sigma)$ is a variable assignment for f. Based thereon, σ *satisfies* c, written $\sigma \models c$, if and only if substitution of the variables in f according to $\alpha(\sigma)$ leads to a true sentence. An example linear constraint for open net D in Fig. 1 is $\mathsf{accept} + 2 \cdot \mathsf{reject} \leq 2$ which is satisfied by any transition sequence with $\alpha(\sigma)(\mathsf{accept}) = 0$ and $\alpha(\sigma)(\mathsf{reject}) = 1$. We call c *satisfiable* if there exists some σ satisfying c. We combine such linear constraints with the boolean operators $\vee, \wedge, \neg$ to *formulas* with the obvious semantics. Example formulas are $\varphi_1 = \mathsf{accept} \leq 1 \wedge \mathsf{accept} + \mathsf{reject} \leq 1$ and $\varphi_2 = \mathsf{accept} \leq 1 \wedge -\mathsf{accept} \leq -2$. Any transition sequence σ with $\alpha(\sigma)(\mathsf{accept}) = 0$ and $\alpha(\sigma)(\mathsf{reject}) = 1$ satisfies φ_1, but φ_2 is not satisfiable.

A formula φ can be used to describe the channel usage of an open net N: Each terminating firing sequence $\sigma \in \mathcal{L}(N)$ needs to satisfy φ. We call such a formula *fingerprint* of N. Example fingerprints for D and S in Fig. 1 are $\varphi_{\mathsf{D}} = \mathsf{document} - (\mathsf{feedback} + \mathsf{accept} + \mathsf{reject}) = 0 \wedge \mathsf{accept} + \mathsf{reject} = 1$ and $\varphi_{\mathsf{S}} = \mathsf{document} - \mathsf{feedback} = 0$.

Services may have infinitely many fingerprints. We can classify a fingerprint based on the formal sums that appear on left hand sides in its constraints: A fingerprint containing only formal sums from a set F is called F-fingerprint. Fingerprint φ_{D} is a $\{\mathsf{document} - (\mathsf{feedback} + \mathsf{accept} + \mathsf{reject}), \mathsf{accept} + \mathsf{reject}\}$-fingerprint. The idea is to *compute* a F-fingerprint for a given set of formal sums F. Our algorithm yields a fingerprint of the form $\bigvee_{m \in \Omega} \bigwedge_{f \in F}((f \geq l_m^f) \wedge (f \leq u_m^f))$: Each conjunctive clause represents the firing sequences resulting in a specific final marking m by specifying two inequalities per formal sum f. We sketch our algorithm for computing l_m^f and u_m^f for some given f and m: The set of firing sequences resulting in a specific final marking m is over-approximated by a system of linear equations, called the *state equation* for m [10]. The state equation is derived from the structure of N, its solutions are vectors over transition variables, which we transform to channel variables. This yields a valid base to minimize and maximize a formal sum of channel variables, resulting in valid l_m^S and u_m^S. If the solver returns *unbounded*, we skip this result. If the solver returns *infeasible*, we skip this final marking. This may result in an empty fingerprint: N might not be compatible to any partner. The complexity for computing such a F-fingerprint of N is that of solving $2 \cdot |\Omega| \cdot |F|$ integer linear programs. In [1], we discuss different generic parameter sets, i.e. sets of formal sums that can be canonically derived from an interface or an open net structure. As a rule of thumb, the complexity of the formal sums has an impact on the precision of the fingerprint: φ_{D} is more precise than $\varphi'_{\mathsf{D}} = \mathsf{document} - (\mathsf{feedback} + \mathsf{accept} + \mathsf{reject}) = 0$.

Matching fingerprints φ_N and φ_Q is a method to semi-decide compatibility of N and Q by deciding satisfiability of $\varphi_N \wedge \varphi_Q$. We assume that φ_N and φ_Q have the above introduced structure. We transform $\varphi_N \wedge \varphi_Q$ into an equivalent formula in disjunctive normal form by applying the distributivity rule. We then check satisfiability for each conjunction which is equivalent to solving a system of linear inequalities. As soon as one satisfiable conjunction is found, we terminate the matching process with *inconclusive*, otherwise with *incompatible*. Different variants of matching, for example matching a φ_N directly with Q, are discussed in [1]. For example, taking the fingerprints φ_{D} and φ_{S}, we find that $\varphi_{\mathsf{D}} \wedge \varphi_{\mathsf{S}}$ is unsatisfiable and thus N and Q are incompatible. Turning to φ'_{D}, we notice that $\varphi'_{\mathsf{D}} \wedge \varphi_{\mathsf{S}}$ is satisfiable, which proves that matching is not a sufficient condition.

4 Case study

For validating our approach, we had to build up a large number of services. As a sufficiently large set of actual services was not available to us, we generated "close to real" services as follows. We started with a large set of *real* industrial business processes available to us. They have been modeled using the IBM WebSphere Business Modeler, then anonymized so they could be made available to us and finally translated into anonymized Petri net models. The used set of business processes has been analyzed

Table 1. Testbed statistics and analysis results

Library	# Processes	# Composites	# Weakly terminating	# Matching inconclusive
A	127	2412	252	672
B1	43	2066	20	597
B2	49	592	25	209
B3	105	3460	210	1165

Table 2. State-based approach vs. fingerprint approach

Library	**State-based**	**Fingerprint**			
	Total	Computation	Matching	Model checking	Total
A	>48h	2m49s	30s	28h	≈28h
B1	18m3s	2m57s	29s	6m9s	6m38s
B2	30m43s	53s	12s	16s	28s
B3	>36h	11m6s	1m29s	2h	≈2h

in [11]. We decomposed the processes into asynchronously communicating services following the idea of [6,12] using Diane. Due to different decomposition options, we obtain a rather big set of services. Two services that each have been obtained from a weakly terminating business process model are not necessarily compatible. In addition, several original models have not been weakly terminating in the first place. The original process set is organized into libraries, each containing models from similar business fields, we thus experimented with one sample set for each library. As the final preparation, we paired interface matching services inside the same sample set. In a first run, we model checked each composite with LoLA. In a second run, we computed the fingerprints of the components with Linda and matched them with Yasmina. Table 1 lists statistics of the used testbed and analysis results: For each library, we list the number of processes and service pairs, i.e. composites. The fourth column shows how many of those composites were actually weakly terminating, while the fifth displays for how many composites the fingerprint matching of the components returned inconclusive.

In Table 2, we compare the run times of a pure state based approach with the run times of the proposed fingerprint based approach. To this end, we checked all pairs of partner services that stem from the same library. The reported times are the overall times for executing all these checks within a library: The second column lists total amount of time for model checking the composites. The third column states the run time of the fingerprint computation. The fourth column displays the time needed for fingerprint matching. For all inconclusive results, we used model checking, resulting in run times as given in the fifth column. Finally, the sixth column displays the total amount for the fingerprint based compatibility check, which does not include the run time for fingerprint computation.

We see that for about two thirds of the individual problems, the fingerprint check tells that these services are incompatible. These are the problem instances for which it is not necessary to perform a subsequent model checking. For the remaining services, model checking must be applied in any case. Hence, the speed-up can be seen in comparing the overall time of model checking *all instances* with the overall time of all fingerprint matchings plus the overall time for those model checking runs where the fingerprint check was inconclusive. The runtime of the fingerprint matching alone does not contribute significantly to the overall run time and the fingerprint approach requires only about one third of the state-based approach. The used tools Diane, LoLA, Linda, and Yasmina are open source tools available at `http://service-technology.org/tools`.

5 Conclusion

In this paper we have considered service communication fingerprints as an approach for pre-selecting appropriate interaction partners with respect to weak termination. We used the state equation of the underlying Petri net to derive constraints over the set of synchronous and asynchronous message event occurrences. Communication fingerprints are considerably small in comparison to the state space of a service. We considered a simple and efficient procedure for obtaining a suitable (not necessarily optimal) communication fingerprint. Matching fingerprints amounts to solving linear programming problems. Our experiments show that the fingerprint approach can significantly speed up service discovery. Our approach is complementary to testing observed behavior against model behavior using frequency profiles [13] and keeping repositories of behavioral profiles [14,15]. Both approaches apply to monolithic workflow and are restricted to transition occurrences. Our approach is different from compositional analysis of invariants of functional nets [6]: We analyze communication patterns which are inherently related to communication.

For future work, we shall consider the application of fingerprints in the synthesis of livelock-free partners. Further, we shall experiment how service communication fingerprint registries created to store subclasses of potentially compatible partners contributes to speeding up operations on behavioral registry [16].

Acknowledgments. Olivia Oanea and Karsten Wolf are supported by the German Research Foundation (DFG) under grant WO 1466/11-1.

References

1. Oanea, O., Sürmeli, J., Wolf, K.: Service discovery using communication fingerprints. Informatik-Berichte 236, Humboldt-Universität zu Berlin (2010)
2. Clarke, E.M., Peled, D., Grumberg, O.: Model Checking. MIT Press, Cambridge (1999)
3. Wolf, K., Stahl, C., Ott, J., Danitz, R.: Verifying livelock freedom in an SOA scenario. In: ACSD 2009, pp. 168–177. IEEE, Los Alamitos (2009)
4. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In: Dumas, M., Heckel, R. (eds.) WS-FM 2007. LNCS, vol. 4937, pp. 77–91. Springer, Heidelberg (2008)
5. Alves, A., et al.: Web Services Business Process Execution Language Version 2.0. Technical Report CS-02-08, OASIS (2007)

6. Zaitsev, D.A.: Compositional analysis of Petri nets. Cybernetics and Systems Analysis 42(1), 126–136 (2006)
7. Sürmeli, J.: Profiling services with static analysis. In: AWPN 2009 Proceedings, of CEUR Workshop Proceedings, vol. 501, pp. 35–40 CEUR-WS.org (2009)
8. Oanea, O., Wolf, K.: An efficient necessary condition for compatibility. In: ZEUS, of CEUR Workshop Proceedings, vol. 438, pp. 81–87 CEUR-WS.org (2009)
9. Wolf, K.: Does my service have partners? In: Jensen, K., van der Aalst, W.M.P. (eds.) Transactions on Petri Nets. LNCS, vol. 5460, pp. 152–171. Springer, Heidelberg (2009)
10. Lautenbach, K.: Liveness in Petri Nets. St. Augustin: Gesellschaft für Mathematik und Datenverarbeitung Bonn, Interner Bericht ISF-75-02.1 (1975)
11. Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Instantaneous soundness checking of industrial business process models. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, Springer, Heidelberg (2009)
12. Mennicke, S., Oanea, O., Wolf, K.: Decomposition into open nets. In: AWPN 2009, Proceedings of CEUR Workshop, pp. 29–34 CEUR-WS.org (2009)
13. van der Aalst, W.M.P.: Matching observed behavior and modeled behavior: an approach based on Petri nets and integer programming. Decis. Support Syst. 42(3), 1843–1859 (2006)
14. Weidlich, M., Weske, M., Mendling, J.: Change propagation in process models using behavioural profiles. In: SCC 20209, pp. 33–40. IEEE, Los Alamitos (2009)
15. Weidlich, M., Polyvyanyy, A., Mendling, J., Weske, M.: Efficient computation of causal behavioural profiles using structural decomposition. In: Lilius, J., Penczek, W. (eds.) PETRI NETS 2010. LNCS, vol. 6128, Springer, Heidelberg (2010)
16. Kaschner, K., Wolf, K.: Set algebra for service behavior: Applications and constructions. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 193–210. Springer, Heidelberg (2009)