Profiling Services with Static Analysis

Jan Sürmeli

Humboldt-Universität zu Berlin, Institut für Informatik
Unter den Linden 6, 10099 Berlin, Germany
suermeli@informatik.hu-berlin.de

Abstract. In a service-oriented architecture, *Services* are components that interact with each other via well-defined interfaces. *Open nets* are a special class of Petri nets, designed to model the behavior of open systems. Asynchronous interaction and stateful behavior complicate predicting the combinations of messages that a service can process. We present *profiles* which support the modeler in verifying compliance of the model with given constraints, without knowing its future environment. We explain the computation of profiles by static Petri net analysis.

1 Introduction

In a service-oriented architecture (SOA) [1], *services* interact with each other by exchanging messages over predefined *channels*. Typically, services are understood as software artifacts that offer a functionality over a well-defined *interface*, defining those channels a service uses. Control and data flow of services heavily depend on the interaction with other services. We aim at analyzing the behavior of a service S and thus model S with *open nets* (or service nets, open workflow nets) [2]. We will describe this extension to classical Petri nets in Sec. 2.

Well-developed methods to analyze the behavior of a service S with open nets, already exist alongside different *behavioral correctness criteria* such as *controllability* [3] and *exchangeability* [4]. Those criteria can be verified by *dynamic* techniques and were implemented in a tool chain[1]. Another subject is to check *behavioral constraints* on services and their compositions. We can demand *orders* on messages, *causalities*, *limits* et cetera. Work in this area has been done in [5]. We can think of different levels of abstraction as well as extensions of the classical evaluation of a constraint to true or false, e.g. three valued logics, probabilities or costs. On top of that, services behave dependent on their environment, so we are interested in overapproximating the behavior for all or a subclass of all possible environments. Work on structural analysis on open nets has been done in [6].

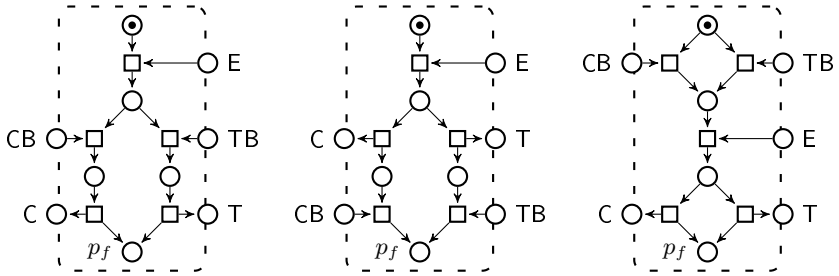
In this paper we present an approach to solve a specific class of behavioral constraints based on well-known *static* Petri net analysis techniques. Such a constraint specifies lower and upper *bounds* for the occurrence of events in the interaction with other services. An *event* occurs when a message is sent or received. A constraint might e.g. demand the exchange of a single message or

[1] Available at http://www.service-technology.org/tools

restrict the occurrence of a *linear combination* of events. A *profile* for S is a set of constraints that S satisfies. Since there exists an infinite number of linear combinations of events, there also exists an infinite number of profiles for S, providing different levels of precision. But we can influence this aspect when computing a profile, such that it meets the requirements of the use case. We formally define the class of constraints, the concept of profiles as well as their computation with static Petri net methods in Sec. 3.

Profiles can be applied in different phases in the lifecycle of a service S. The modeler of S can use profiles to prove that the model complies with the specification. Other analysis methods can benefit from profiles, in this case profiles are used as a preprocessor. During implementation of S, profiles can be used to generate test cases, since a profile contains constraints that the model satisfies. After deploying S it will be available in a service repository. A profile can be used to store an abstraction of S that can support behavioral query resolving. A profile, however, does not cover all aspects of S: It is restricted to abstract interaction behavior. We will discuss the application of profiles in Sec. 4. We conclude the paper in Sec. 5 with open issues and ideas for further work.

2 Modeling with Open Nets



(a) N_1 (b) N_2 (c) N_3

Fig. 1. Three open nets with the final marking $[p_f]$

Two services interact by exchanging messages over *channels* predefined in the interface of a service. We assume an *asynchronous* communication model: Sending and receiving of a message does not occur in the same moment, as opposed to hand-shaking-techniques. Thus, for each exchanged message, two events occur: Sending and receiving. From the viewpoint of one of the involved services however, only one of the two events is observable, namely the event of sending or receiving a message by the service itself. For example, service S sends a message that is later received by service S', then for S the sending event is observable and from the viewpoint of S' only the receiving event occurs.

Furthermore we assume that a service only communicates unidirectional over a message channel: Either messages are sent or received via this channel.

We use the classical syntax and semantics of Petri nets as in [7]. We define the behavior of a Petri net N as the set of all sequential runs in N. *Open nets* are Petri nets that are augmented by a *final marking* and an *interface* for message exchange, the latter is realized by designating some places as input and some as output places. These so called *interface places* are used as connectors for the composition. Other places are called *internal places* and the net structure is that of a classical Petri net.

Figure 1 shows three open nets with the same interface, graphically emphasized by the dashed line. E, TB and CB are input places, C and T are output places. N_1, N_2 and N_3 are models of simple coffee/tea machines that have one button for coffee (CB), one for tea (TB), an input for money (E) and two output slots, one for coffee (C) and one for tea (T). Although the three models have the same interface, they differ obviously in the internal structure.

The modeling of interface places is only the prerequisite for message exchange, message exchange can only occur between a number of open nets that are *composed*. Two two open nets N and N' can be composed if they are *syntactically compatible*, meaning that they do not share internal places or transitions and have compatible interfaces. Two interfaces A and B are compatible if the output (input) channels of A are not used as output (input) channels in B and vice versa. *Composition* is done by union of the net elements, composing initial and final marking and merging identically named interface places that become internal places.

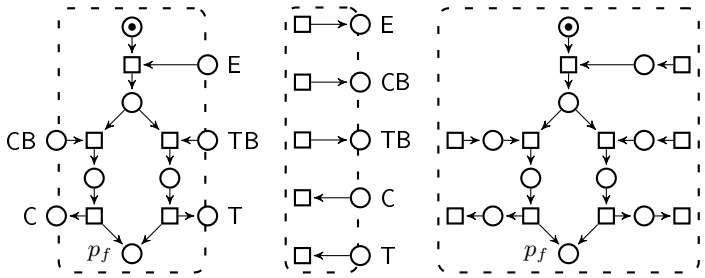
3 Profiles for Open Nets

In this paper, we approach a specific class of constraints: *Lower and upper bounds for event occurrence*. Intuitively a constraint of that class specifies the bounds for legal interaction: An event or a linear combination of several events is only allowed to occur in those bounds. An example for a constraint is $2 \leq a \leq 7$, meaning that the event a occurs at least two and at most seven times. A constraint however need not give an integer value for one or both bounds, it can also specify one of the bounds as *unbounded*, meaning that interaction is not constrained in that direction. For example $2 \leq a \leq \_$ demands that event a occurs at least two times but might occur infinitely often. Dependencies between messages can be expressed easily as well: $1 \leq a+b \leq 1$ states that always exactly one of the two events occur. $0 \leq a - b \leq 0$ demands that a and b occur equally often, since the inequality can easily be transformed to $a = b$. We can not specify temporal orders between messages, causalities or more complex dependencies. The open nets N_1, N_2, N_3 in Fig. 1 all three comply to the constraints $E = 1$, $0 \leq C \leq 1$, $CB + TB = 1$. Only N_1 and N_2 comply with the constraints $CB - C = 0$, $TB - T = 0$, $CB + T = 1$ and $TB + C = 1$ that demand that TB (CB) only occurs with T (C). We can not express order restrictions with such constraints, so we can not forbid a behavior as in N_2.

Formally, a *constraint* c is a quadruple $\langle A, \theta, l, u \rangle$, where A is a finite set of events, $\theta : A \to \mathbb{Z}$ is a linear combination of events and $l, u \in \mathbb{Z} \cup \{\_\}$. The semantics of such a constraint is as follows: θ, l, u form an inequality ϕ of the form $l \leq \theta \leq u$, with the elements of A as variables. Let N and N' be open nets. A run r in $N \oplus N'$ satisfies c from the viewpoint of N, written $c \vdash_N r$ if and only if the occurrence rate of events in the projection of r to transitions of N is a satisfying assignment for the variables in c. A sequential run is *terminating* if and only if it starts at the initial marking and ends at the final marking. N complies with a constraint c if, for an arbitrary N' and every terminating sequential run r in $N \oplus N'$, $c \vdash_N r$ holds.

A *profile* of N specifies a set of constraints that N complies with. These constraints need not be the strictest ones that apply but might be quite liberal. Computation of a profile for an open net can be done by *static* analysis, avoiding state space construction. Thus, a profile is an *abstraction* of the behavior of N with any arbitrary service N'.

Knowledge of N' can not be assumed, therein we find the first challenge for computing a profile. We approach the problem by overapproximating every possible N' by a canonical open net that has the most liberal interaction behavior, called the *unrestricted environment* of N, denoted as $\widehat{N}$: An open net, only consisting of exactly one transition for each interface place of N and no internal places. One would not encounter $\widehat{N}$ in the practical field, but it proves to be very helpful in analyzing the service in interaction with $\widehat{N}$ and deducing its behavior in an arbitrary environment from the results. The connection between the behavior of $N \oplus \widehat{N}$ and $N \oplus N'$ is the following: Fixing any run in $N \oplus N'$, we can find a run $N \oplus \widehat{N}$, such that the two are equivalent if we just concentrate on N's part. We demonstrate this on the example of the open net N_1 in Fig. 2(a): Figure 2(b) shows the unrestricted environment for N_1 and their composition is depicted in Fig. 2(c).



(a) N_1 (b) $\widehat{N_1}$ (c) $N_1 \oplus \widehat{N_1}$

Fig. 2. N_1 (repeated from Fig. 1), its unrestricted environment $\widehat{N_1}$ and $N_1 \oplus \widehat{N_1}$.

However, for computing a profile we do not construct and explore the state space, but use a classical static Petri net method, namely the *state equation*, a

canonical system of linear equations, taking into account two markings β, β'. For every run from β to β', there exists a solution m, such that transitions occur as often in the run as given by m. We construct the state equation of $N \oplus \widehat{N}$, setting $\beta = \alpha$ and $\beta' = \omega$. We add an equation for each possible event in N, specifying the transitions of N that let an event occur. The set of all solutions is thus both an overapproximation of all terminating sequential runs r in $N \oplus \widehat{N}$ and gives the occurrence rates for events in r from the viewpoint of N.

Given a set of linear combinations of events, we can now apply *linear programming* to find lower and upper bounds for these combinations. Solutions of these linear programs are constraints that N complies with. Thus we directly construct a profile. The set of linear combinations is the only parameter, the rest is canonical on the net structure. We can distinguish between two general starting points for profiles: (1) there exists a specification, given as constraints before computing the profile, (2) we compute the profile in advance. In the first case, the input for the profile computation can be taken directly from the specification. In the second case, the selection of useful linear combinations has to be done manually, although we can imagine taking into account structural properties like invariants, conflict situations and the like.

The computation of profiles for a given open net has been implemented in the tool Linda[2]. It takes a set of constraints as input and computes an according profile, using the lp_solve-library[3] to solve linear problems. Interoperability with other tools is enforced by usage of the same open net format as the other tools in the above mentioned tool chain.

4 Application of Profiles

Given an open net N and a set of constraints C, we can create a profile ψ to determine compliance of N to the constraints in C. In some cases, however, we are neither able to prove compliance nor non-compliance with this method. We demonstrate the compliance checking process with an example. Let N be

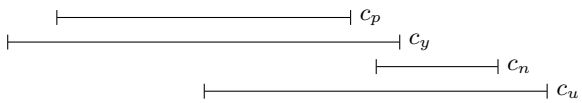


Fig. 3. The bounds given by four constraints c_p, c_y, c_n, c_m.

an arbitrary open net and c_y, c_n, c_u be constraints that restrict the same linear combination. Creating a profile for any of the singleton sets $\{c_y\}$, $\{c_n\}$ and $\{c_u\}$ leads to the same result, we denote it as $\{c_p\}$. Let the bounds given by the constraints c_p, c_y, c_n, c_u be as depicted in Fig. 3. We first check compliance with

[2] Available at http://www.service-technology.org/tools/linda
[3] Available at http://lpsolve.sourceforge.net

c_y: N complies with c_y since the bounds given by the profile imply those of c_y. In contrast to that, N does not comply with c_n: The lower bound of c_n is greater than the upper bound of c_p. In the case of c_u however, we can not decide compliance or non-compliance of N with c_u by the method of profiling.

Given a profile ψ of N, we can also check compliance of N with a constraint c, although the linear combination of c is not directly restricted by any constraint in ψ: We determine a constraint restricting the same linear combination as c that is implied by the constraints in ψ and then do compliance checking as explained above. Finding such an implied constraint can be done by linear programming.

5 Conclusion and Further Work

We have described a class of *constraints* for the interaction behavior of a service. Our approach to the solution is the computation of a *profile*, a set of constraints that a service complies with, using *static* Petri net analysis methods. We have demonstrated how profiling can be used to *check compliance*, answering with *yes*, *no* or *unknown*.

It is part of further work to determine the inputs of profile computation if they can not be derived from a specification, such that a profile can be stored as an abstraction of an open net. Since a profile is an abstraction, we can think of making use of refinement techniques to further explore questions answered with *unknown*. We will embed profiles in different *tools* to analyze services and their composition, so that they benefit from preprocessed information or on-the-fly checking of constraints.

References

1. Gottschalk, K.: Web services architecture overview. http://www.ibm.com/developerworks/web/library/w-ovr/ (2000)
2. Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA. Annals of Mathematics, Computing & Teleinformatics **1**(3) (2005) 35–43
3. Wolf, K.: Does my service have partners? T. Petri Nets and Other Models of Concurrency **2** (2009) 152–171
4. Stahl, C., Massuthe, P., Bretschneider, J.: Deciding Substitutability of Services with Operating Guidelines. Transactions on Petri Nets and Other Models of Concurrency II, Special Issue on Concurrency in Process-Aware Information Systems **2**(5460) (March 2009) 172–191
5. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral constraints for services. In Alonso, G., Dadam, P., Rosemann, M., eds.: Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings. Volume 4714 of Lecture Notes in Computer Science., Springer-Verlag (September 2007) 271–287
6. Oanea, O., Wolf, K.: An efficient necessary condition for compatibility. In Kopp, O., Lohmann, N., eds.: ZEUS. Volume 438 of CEUR Workshop Proceedings., CEUR-WS.org (2009) 81–87
7. Reisig, W.: Petri nets: an introduction. Springer-Verlag New York, Inc., New York, NY, USA (1985)