On synthesizing service behavior that is aware of semantical constraints

Karsten Wolf

Universität Rostock

Abstract. Without taking care of the semantics of messages, every message is an isolated entity that can be created and sent at will. This leads to anomalies like a synthesized service that sends a filled form before having received the empty form. In this paper we pick up ideas from adapter synthesis for taking care of semantical constraints and develop them into two directions. First, we show that the approach taken for adapter synthesis can be applied to synthesis of services in general. Second, we argue that the taken approach is in a certain sense complete.

1 Introduction

We synthesize service behavior for several purposes. First, we can show that it is possible to interact correctly with a service by constructing a fitting service behavior [1–3]. This approach can be used as a sanity check for given services. Second, we can try to exploit the canonicity of the computed behavior and use it for characterizing all correctly interacting partners of a service [4]. Third, we can transform the computed behavior into executable code that can be executed as a particular partner of the service.

An instance of the latter approach is the synthesis of a behavioral adapter A between two given services P and R. At first glance, A is not much more than a correctly interacting partner of a disjoint composition of P and R. However, this general setting permits a number of anomalies with arise from the fact that behavioral approaches typically abstract from the content of exchanged messages. In plain language, messages are named a, b, c, $\ldots$ without taking care of whether they denote the submission of a password, an address, a simple acknowledgement, a real item (sold book), or anything else. This simplification leads to anomalies like the synthesis of an adapter that can "invent" a password or forward two copies of a received (real!) book. Having observed this, virtually all approaches to behavioral adapter synthesis [5–11] start with some kind of specification that expresses appropriate constraints for the activities that can be performed on certain messages. Although different in technical detail, the specifications all express more or less the same class of constraints. This class of constraints can thus be considered as mature.

In this note, we demonstrate that the approach taken in adapter synthesis extends to the synthesis of service behavior in general. That is, we can avoid semantic anomalies in any kind of synthesis of services by taking into account an appropriate specification of semantical constraints. For instance, we can suppress synthesis of a service that sends a message containing a session identifier before having received this identifier. We can avoid sending a signed contract before having received the unsigned version thereof. There are many other examples of constraints that are imposed by the semantics of message contents.

As a second contribution, we argue about completeness of our approach. We claim that (under a few technical limitations like boundedness) we can synthesize a correctly interacting partner whenever one exists. To this end, we propose a definition for "arbitrary partner that respects given semantical constraints" and then show our completeness result using this definition.

2 Services

We model services as open nets, i.e. as place/transition Petri nets with an initial marking, a set of final markings, and a set of places serving as interface. Initial and final markings have no tokens on interface places. We require further that final markings do not enable any transition (although transition may become enabled by putting tokens on input places). The interface is divided into input and output places and we require that no transition takes tokens from an output place an no transitions puts tokens on an input place.

There exist translations from industrial languages like WS-BPEL into open nets [12] and vice versa [13] which proves the suitability of open nets for modeling services. Open nets are composed by merging equally named interface places (an input place of one service with an output place of the other one). The merged places are then removed from the interface. Initial and final markings are composed canonically (remember that we require them to have no tokens on interface places). We denote the composition of two open nets P and R by $P \oplus R$.

Composition may lead to an open net with empty interface which we call closed net.

A closed net is deadlock-free iff all markings without enabled transitions are final. An open net P is controllable iff there exists an open net R such that their composition $P \oplus R$ is a deadlock-free closed net.

It is just one possibility to use deadlock freedom as the underlying property for controllability. Other requirements could include livelock freedom or any other desired property. Deadlock freedom is, however, the most prominent property discussed in the context of web services.

3 Synthesis of service behavior

Controllability of an open net P is decidable [1, 2] if two conditions are satisfied. First, the inner of P (the net obtained by removing the interface of P) must be bounded and second, we restrict the set of considered partners to those R where the composition $P \oplus R$ yields k-bounded (now merged) interface places, for some a priori given k. In effect, the composition of considered nets are finite state systems. In absence of the first restriction, controllability becomes undecidable [14] even in presence of the second one. If only the second condition is dropped, decidability of controllability is unknown.

In presence of the mentioned conditions, controllability can be decided by synthesizing (the state space of) a canonical R as required by the definition of controllability. The resulting state space (a kind of automaton) can be transformed into a Petri net using standard approaches [15–17] and further into languages like WS-BPEL [13]. In the resulting WS-BPEL process, transitions of the Petri net appear as opaque activities. Refining these activities, one obtains an executable WS-BPEL process. We skip details as they are not necessary for understanding the results in this note.

The synthesized partner provides a communication skeleton for correct interaction with P and is thus valuable beyond witnessing controllability. If one desires to invoke P, he can automatically generate the corresponding code from the description of P. If one does not want to use P arbitrarily, additional constraints may be applied using the techniques of [18].

A particular application of this approach is the automated synthesis of an adapter A between two services P and R. If the composition of P and R is not deadlock-free, an intermediate component may mediate the communication between the two and enforce deadlock freedom. Formally, A is a service such that $P \oplus A \oplus R$ is deadlock-free and can thus be synthesized as a witness for controllability of $P \oplus R$. More precisely, we need to rename interfaces of P and R such that they become disjoint. This way, all communication between P and R will pass A.

4 Semantical constraints

As we synthesize behavior, we are not necessarily interested in the details of the semantics of exchanged messages as such. We are only interested in the impact of semantics on behavioral issues. Experience from adapter synthesis suggests that the main impact of semantical issues is to constrain the ability of a service to manipulate message contents. The semantics determines

Table 1. Examples of semantical constraints in terms of transformation rules

Constraint	Rule	Example pro	Example con
Create a	$\mapsto a$	own password, simple acknowledgement	foreign password
Copy a	$a \mapsto a, a$	address	money transfer, transaction number
Delete a	$a \mapsto$	electronic message	real item (e.g., book)
Transform a into b	$a \mapsto b$ $a \mapsto a, b$	length in feet to length in meters	zipcode to length
Split a into b, c, d	$a \mapsto b, c, d$ $a \mapsto a, b, c, d$	address to name, city, street	the other way round
Merge a, b, c into d	$a, b, c \mapsto d$ $a, b, c \mapsto a, b, c, d$	name, city, street to address	the other way round
Recombine a, b to c, d, e	$a, b \mapsto c, d, e$ $a, b \mapsto a, b, c, d, e$	at reader's discretion	

whether or not the content of a message can be generated, copied, deleted, or computed from the content of other messages.

In [11], we proposed to specify semantical constraints as a set of transformation rules. Each rule consists of two bags saying that the contents of the right hand side messages can be determined from the contents of the left hand side messages, thereby consuming the messages at left hand side. Consumption of involved messages makes sense as real items may be involved while non-consumption of a message may be modeled by re-generating it on the right hand side. For convenience, the universe used for building bags is a set of *semantical entities* that contains but is not restricted to the names of exchanged messages. This way, we have more freedom to model dependencies.

Table 1 lists those semantical constraints which have been proposed in the context of adapter synthesis, together with examples where they make sense as well as examples where they don't make sense. The examples show that the applicability of a rule indeed depends on the semantics of the message contents and cannot be inferred from the service protocol. Consequently, we consider a scenario where the constraint specification is part of the input to the synthesis problem.

There are various ways to generate a specification of semantical constraints. First, they may be generated manually. Since the transformation rules are rather simple, this should not be a problem. Second, they may be inferred using semantic web technology like ontology reasoning. State of the art in this field is beyond the scope of this note. Third, they may become part of the service construction process using some (may be intra-organizational) modeling standard.

5 Synthesis of service behavior in presence of semantical constraints

Our approach (already exercised in [11]) consists of the following steps. Given an open net P and a specification C of semantical constraints, we transform C into an open net S that covers the whole interface of P. S basically manages the message transfer from and to P as well as the transformation of semantical entities according to C. Via a separate interface, it is possible to trigger any activity in S and receive a notification of its execution. In a second step, we synthesize a correctly interacting partner R for $P \oplus S$ using the traditional approach (i.e., not taking care of semantics). Finally, we merge R with S into the final result which can be further optimised using Petri net reduction rules and, if desired, transformed into WS-BPEL.

In this agenda, the construction of S is obviously the crucial part as all other steps rely on existing technology. Consider some given service P and a set C of semantical constraints ranging on a set E of semantical entities. Let I and O be the sets of input and output places of P.

We assume that $I \cap O = \emptyset$ and $I \cup O \subseteq E$. For simplicity of presentation, we assume that for each rule in S, both sides are sets; the general case follows analogously using Petri nets with arc multiplicities.

For defining the service S, we use names from the space $(E \cup C) \times \{e, n, c, r, s\}$, where e, n, c, r, s denote characters instead of variables; hence we assume that these names do not occur in the given service.

The interface of S consists of output places I, input places O (i.e., the interface of P in opposite orientation), and some input and output places specified below. For each entity $e : e \in E$, we introduce in service S an internal place (e, c) (c for "copy"). In the initial and final markings, the internal places are empty, although this can easily be generalized in future work.

Service S has three kinds of transitions. For every input place $o : o \in O$, there is a transition (o, r) (r for "receive") to move arriving messages from interface place o to their internal place (o, c). For every transformation rule $w : w \in C$, there is a transition (w, c) to perform the actual transformation in terms of the internal places. Finally, for every output place $i : i \in I$, there is a transition (i, s) (s for "send") to move messages from their internal place (i, c) to interface place i.

Finally, we discuss the additional interface places for the controller. For every input place $o : o \in O$, output place (o, n) (n for "notify") notifies an arrived message o. For every transformation rule $w : w \in C$, input place (w, e) (e for enable) enables transformation rule w, and output place (w, n) notifies an execution of w. Finally, for every output place $i : i \in I$, input place (i, e) enables the delivery of a message i (once available).

Definition 1 (Service S). *Let I, O, E, C be as introduced before. The corresponding service S is defined as an open net with the following constituents:*

$$\begin{aligned}
P &= (E \times \{c\}) \;\cup\; P_i \;\cup\; P_o \\
P_i &= O \;\cup\; (C \times \{e\}) \;\cup\; (I \times \{e\}) \\
P_o &= I \;\cup\; (C \times \{n\}) \;\cup\; (O \times \{n\}) \\
T &= (O \times \{r\}) \;\cup\; (C \times \{c\}) \;\cup\; (I \times \{s\}) \\
F &= F_r \cup F_c \cup F_s \\
F_r &= \textstyle\bigcup_{o \in O} \;\{\; [o, (o, r)],\; [(o, r), (o, n)],\; [(o, r), (o, c)] \;\} \\
F_c &= \textstyle\bigcup_{w = X \mapsto Y \in C} \;(\; \{[(m, c), (w, c)] \mid m : \; m \in X\} \;\cup \\
&\qquad \{[(w, e), (w, c)],\; [(w, c), (w, n)]\} \;\cup\; \{[(w, c), (m, c)] \mid m : \; m \in Y\} \;) \\
F_s &= \textstyle\bigcup_{i \in I} \;\{\; [(i, c), (i, s)],\; [(i, e), (i, s)],\; [(i, s), i] \;\} \\
m_0 &= \underline{0} \\
\Omega &= \{\underline{0}\}
\end{aligned}$$

We use $\underline{0}$ to denote the marking that is zero in every place. By construction, all outputs to P have been obtained from the input of P using the transformation rules only. The actual scheduling of rule applications and message deliveries is left to a controller using the remaining interface.

The inner of S may be unbounded which would complicate further synthesis. For this reason, we pragmatically introduce some capacity on the places of S that, if chosen sufficiently large, should not restrict our results undueley.

Having generated S, it remains to synthesize a partner R of $P \oplus S$ which can be done using the approach of [1, 2]. R basically schedules the application of available actions: it triggers the application of transformations as well as the shipment of messages. Its decisions are based on notifications about incoming messages and applied transformations.

A WS-BPEL process constructed from $R \oplus S$ would contain an opaque activity for each transition, including those that represent the application of transformation rules. In several situations, it is possible to complement the specification of semantical constraints with code snippets that actually implement the specified transformation. In these cases, we may end up with an executable WS-BPEL process that implements the whole interaction with P and is correct by construction.

6 Obeying semantical constraints

In the next section, we wish to establish a result of the following kind: Given a service P and a set C of semantical constraints, if there is any R such that R interacts correctly with P and R obeys the semantical constraints, then $P \oplus S$ is controllable. A result of that kind is only valuable if the definition of "to obey the semantical constraints" is as liberal as possible. In this section, we propose such a definition. For simplicity, we consider only nets where all arc multiplicities are equal to one.

As a starting point, we assume that R has one place for each semantical entity occurring in C. This may be seen as a restriction. Since, however, typical semantical entities are exchanged messages for which there is anyway a representing place, this condition should not be too restrictive. Let P_S be the set of places that represent semantical entities. Let F_S be the set of edges that have their source or sink node in P_S.

The idea of our definition is to mark the application of transformation rules in the normal control flow of an open net. To this end, we use some infinite set U. Each element of U represents the application of a single rule in C. There may be several elements in U that represent the same rule. Elements of U are assigned to those edges which are connected with P_S, i.e. we consider a mapping $\psi : F_S \to U$. This way, access to semantical entities is grouped. u represents a rule $X \mapsto Y$ iff the source places of arcs labeled with u match X, the sink places of arcs labeled u match Y, and each consumption activity causally precedes each production activity. Formally, the first requirements amount to $X = \{p \mid [p,t] \in F_S,\ \psi([p,t]) = u\}$, $Y = \{p \mid [t,p] \in F_S,\ \psi([p,t]) = u\}$. Causal precedence is difficult to formalize as we do not want to rule out open nets with cycles. Therefore, we need to separate different instances of transitions which contribute to a rule. This leads to the second restriction. We require that, for each u, every run of the inner of R can be divided into sequentially arrangeable parts such that each part contains exactly one occurrence of each transition contributing to u (i.e. there is a p such that $\psi([p,t]) = u$ or $\psi([t,p]) = u$). Within each part, we may now require that $\psi([p,t]) = u$ and $\psi([t,p]) = u$ implies that t causally precedes or is equal to t' which formalizes the idea that consumption precedes production.

We say that R obeys C iff a mapping ψ with the discussed properties exists.

It is easy to see that $R \oplus S$ as computed in the previous section obeys C.

7 Completeness

With the definition of the previous section we are now ready to claim completeness of our approach.

Theorem 1. *Consider an open net P and a set C of semantical constraints. If P has a correctly interacting partner that obeys C then $P \oplus S$ is controllable where S is the open net constructed from C as described earlier in this note.*

For proving this theorem, let R be a correctly interacting partner of P that obeys C. We transform R into a correctly interacting partner of $P \oplus S$ using the following ideas.

- Rename input places p of R to (p,n) and output places to (p,e). This way, R talks to S instead of P.
- For each used $u \in U$, introduce new places p_u and q_u. These places control the invocation of rules.

– For each $[p, t]$ with $\psi([p, t]) = u$, an arc $[t, p_u]$; for each $[t, p]$ with $\psi([t, p]) = u$, an arc $[q_u, t]$;
– For each u (where c=$X \mapsto Y$ is the rule represented by u), a transition that consumes $|X|$ tokens from p_u and puts one token on (c, e) as well as a transition that consumes one token from (c, n) and puts $|Y|$ tokens on q_u.

By this construction, a rule is invoked in S after having consumed the corresponding semantical entities in R but before having produced any entity. By our requirements on causal dependencies, the construction does not influence the behavior of R. Thus, the resulting partner interacts correctly with $P \oplus S$.

8 Conclusion

We have shown that the approach of [11] applies to partner synthesis in general. We have further shown that, for a quite liberal definition of "obeying semantical constraints" our approach is complete in the sense that we can synthesize a partner that obeys the constraints iff one exists. This result is, of course, subject to the following shortcomings: First, we are restricted to finite state partners with a given bound on the access of interface places. Second, we have artificially limited the concurrent application of rules and intermediate storage of semantical entities in S to make S finite state as well. Third, technicalities in the definition of "obey C" may be further relaxed. Nevertheless, the completeness result should add confidence into our approach.

References

1. Schmidt, K.: Controllability of open workflow nets. In: Enterprise Modelling and Information Systems Architectures. Volume P-75 of LNI. (2005) 236–249
2. Weinberg, D.: Analyse der Bedienbarkeit. Diplomarbeit, Humboldt-Universität zu Berlin (2004)
3. Moser, S., Martens, A., Häbich, M., Müller, J.: A hybrid approach for generating compatible WS-BPEL partner processes. In: Proc. BPM. Volume 4102 of LNCS., Springer (2006) 458–464
4. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In: Proc. ICATPN. Volume 4546 of LNCS. (2007) 321–341
5. Cimpian, E., Mocan, A.: WSMX process mediation based on choreographies. In: Proc. BPM Workshops. Volume 3812 of LNCS. (2005) 130–143
6. Benatallah, B., Casati, F., Grigori, D., Motahari Nezhad, H.R., Toumani, F.: Developing adapters for Web services integration. In: Proc. CAiSE. Volume 3520 of LNCS. (2005) 415–429
7. Dumas, M., Spork, M., Wang, K.: Adapt or perish: Algebra and visual notation for service interface adaptation. In: Proc. BPM. Volume 4102 of LNCS., Springer (2006) 65–80
8. Brogi, A., Popescu, R.: Automated generation of BPEL adapters. In: Proc. ICSOC. Volume 4294 of LNCS. (2006) 27–39
9. Bracciali, A., Brogi, A., Canal, C.: A formal approach to component adaptation. Journal of Systems and Software **74**(1) (2005) 45–54
10. Motahari Nezhad, H.R., Benatallah, B., Martens, A., Curbera, F., Casati, F.: Semi-automated adaptation of service interactions. In: Proc. WWW. (2007) 993–1002
11. 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)
12. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In: Proc. WS-FM. Volume 4937 of LNCS. (2007) 77–91
13. Lohmann, N., Kleine, J.: Fully-automatic translation of open workflow net models into human-readable abstract BPEL processes. In: Proc. Modellierung. Volume P-127 of LNI. (2008) 57–72
14. Massuthe, P., Serebrenik, A., Sidorova, N., Wolf, K.: Can I find a partner? Undecidablity of partner existence for open nets. Inf. Process. Lett. (2008) (Accepted for publication).
15. Ehrenfeucht, A., Rozenberg, G.: Partial 2-structures. Acta Informatica **27** (1990) 315–368
16. Desel, J., Reisig, W.: The synthesis problem of Petri nets. Acta Informatica **33** (1996) 297–315
17. Badouel, E., Darondeau, P.: Theory of regions. In: Lectures on Petri Nets I: Basic Models. Volume 1491 of LNCS., Springer-Verlag (1996) 529–586
18. Lohmann, N., Massuthe, P., Wolf, K.: Behavioral constraints for services. In: Business Process Management 2007. Volume 4714 of LNCS. (2007) 271–287