

Extending the Compatibility Notion for Abstract WS-BPEL Processes

Dieter König¹, Niels Lohmann², Simon Moser¹,
Christian Stahl³, and Karsten Wolf²

¹ IBM Böblingen Laboratory
Schönaicher Straße 220, 71032 Böblingen, Germany
{dieterkoenig, smoser}@de.ibm.com

² Universität Rostock, Institut für Informatik
18051 Rostock, Germany
{niels.lohmann, karsten.wolf}@uni-rostock.de

³ Humboldt-Universität zu Berlin, Institut für Informatik
Unter den Linden 6, 10099 Berlin, Germany
stahl@informatik.hu-berlin.de

Abstract. WS-BPEL defines a standard for executable business processes. Executable processes are business processes which can be automated through an IT infrastructure. The WS-BPEL specification also introduces the concept of *abstract processes*: In contrast to their executable siblings, abstract processes are not executable and can have parts where business logic is disguised. Nevertheless, the WS-BPEL specification introduces a notion of *compatibility* between such an under-specified abstract process and a fully specified executable one. Basically, this compatibility notion defines a set of syntactical rules that can be augmented or restricted by *profiles*. So far, there exists two of such profiles: the Abstract Process Profile for Observable Behavior and the Abstract Process Profile for Templates. None of these profiles defines a concept of behavioral equivalence. Therefore, both profiles are too strict with respect to the rules they impose when deciding whether an executable process is compatible to an abstract one. In this paper, we propose a novel profile that extends the existing Abstract Process Profile for Observable Behavior by defining a behavioral relationship. We also show that our novel profile allows for more flexibility when deciding whether an executable and an abstract process are compatible.

Key words: WS-BPEL, Petri nets, Abstract Profile, Compliance

1 Introduction

The Web Services Business Process Execution Language (WS-BPEL, or BPEL for short) offers a standards-based approach to build distributed applications for business-to-business interactions. A BPEL process implements one Web service by specifying the interactions with other Web services. This allows for building

Both use cases for abstract processes have in common that the abstract process definition is regarded as a *specification*, where an executable process can then be seen as one *implementation* thereof. When one artifact serves as a specification and one artifact provides an implementation of that specification, obviously there must be a way to ensure that they are *compatible*. The BPEL specification calls an executable process compatible to an abstract one if “the executable process is one of the executable completions in the set of permitted completions specified by the abstract process profile”. The definition of an executable completion stops at a syntactical level. Additional syntactic rules restrict the set of allowed executable completions are provided by profiles addressing particular use cases. It can easily be imagined that a compatibility notion can be created at a behavioral level—a technique that is not provided by the BPEL specification and thus not applicable to BPEL processes yet. Furthermore, especially the APPOB imposes restrictions that are unnecessary and too strict from a behavioral compatibility point of view. To show this, we will present a formal model for service contracts and will then extend the existing APPOB by defining two relations, a *conformance* and an *behavioral equivalence* relation. To this end, we define a novel abstract process profile, the Abstract Process Profile for Globally Observable Behavior (APPGOB, cf. Fig. 1). This augmented version of the APPOB will then allow for more flexibility when creating an executable process that conforms to its abstract process. For this purpose, we use a theory on contract-based service composition based on Petri nets. It includes a local conformance criterion, an algorithm to decide conformance, and transformation rules to derive an implementation that conforms to a specification.

The paper is structured as follows: Section 2 introduces the existing profiles and their restrictions and gives an example why these profiles are regarded as too strict. Section 3 presents a formal model for services, based on which an augmented, more liberal profile is presented in Sect. 4. Section 5 shows the augmented profile in action by revisiting the initial example. Finally Sect. 6 and 7 sketch related work and conclude the paper.

2 Motivation

As pointed out in the introduction, both existing profiles are too strict on deciding compatibility. In this section, we will introduce the existing profiles in more detail as well as showing an example to illustrate this problem.

2.1 Abstract Processes in BPEL

The BPEL standard introduces the notion of *abstract processes*. Abstract processes are either used to hide language elements of an executable process, or are not yet fully specified. As mentioned before, in one use case abstract processes can serve as a protocol description. While WSDL operations describe stateless interactions, BPEL processes usually implement stateful Web services which require their exposed operations to be invoked in a particular order. Therefore,

in order to interact with a partner, a business process provider must not only provide a description of the stateless WSDL interface, but he also might want to specify the interaction protocol. This interaction protocol can be represented by an abstract process that shows the externally observable behavior of an executable process while hiding other process model elements. In other words, it describes a service contract fulfilled by a corresponding executable process and avoids the disclosure of internal (potentially confidential) business process logic. This is achieved by taking an executable process and making it more coarse grained by omitting everything that does not belong to the actual observable behavior. A partner interacting with that process would then have to adopt to that behavior in order to deduct his own structure. For the partner interaction aspect of this, there also has been work [2–4] on how to (semi-)automatically create such an abstract partner process out of a given executable one.

In a second use case, an abstract process serves as a template for further refinement. To illustrate this, imagine a business analyst capturing and sketching out a business process, recording it as an abstract process by omitting all the technical details, then handing it off to an IT department in order to add details required for the process to become executable which are not relevant for the business. Such an abstract process may have been generated by business-level process modeling tools like IBM’s Websphere Business Integration Modeler or even by transforming UML into abstract BPEL, for example by using special UML stereotypes [5].

The BPEL standard defines the *Common Base* (cf. Fig. 1), describing the syntax of an abstract process and two types of transformations between abstract and executable processes. The first transformation is a replacement of explicitly modeled *opaque* activities of an abstract process by activities of an executable process. The second transformation consists of *inserting* entities of an executable process at specific places in a process model. Reordering of activities, removing of activities, or changing the control flow is never permitted. Based on these transformations, the common base defines two relations between an abstract process and a set of executable processes, the *Executable Completion* and the *Basic Executable Completion*. Beyond these general syntactical transformations, the specification further allows defining additional syntactical restrictions by means of *profiles*. These additional rules syntactically specify a subset of executable processes. As already mentioned, the BPEL specification provides two concrete profiles for the two use cases described above, the APPOB and the APPT (cf. Fig. 1).

In this paper, we will take a closer look at the first profile. The set of executable completions of an abstract process is restricted by this profile such that the externally observable interactions defined in the abstract process are preserved in all of its executable completions. In other words, while creating an executable process, no transformation must be applied that modifies interactions via partner links already defined in the abstract process. According to the APPOB, in abstract processes

- *join conditions* are not allowed to be hidden,

- the *exit* activity must not be inserted,
- only attributes referencing variables and message parts in partner interactions are allowed to be opaque,
- opaque *from-specs* are allowed,
- endpoint references must not be assigned to/from partner links in a way that the interaction behavior across existing partner links is affected,
- the nesting structure of composite activities around any activity in an abstract process remains unchanged (e.g., it is disallowed to insert a loop activity as a new parent of an existing activity),
- the ability to introduce new branches, handlers, links to existing activities, or scoped declarations is substantially restricted, in particular, modifications must be avoided which affect the branching behavior in a way that conflicts with the specifications in the abstract process, and
- the ability to throw new faults is limited to avoid affecting the existing control flow.

Furthermore, new partner links may be added and used in additional communicating activities.

2.2 Example

As an example, consider the abstract BPEL process of Fig. 2. It describes the abstract process of a travel agency that communicates with three parties: a client service, a hotel service, and an airline service.

After receiving a travel request of the client, the travel agency prepares two orders for the hotel and the airline reservation system. As these preparations are highly data-dependent and do not affect the observable behavior of the travel agency, these activities are specified opaquely. Then, the respective orders are forwarded to the hotel and the airline partner which return their respective offer. Finally these offers are merged and sent to the client.

The APPOB specifies a set of derivable executable completions of this abstract process. Being compatible to the abstract process (i.e., the public view) of the travel agency, they have the same observable behavior in common.

The goal of this paper is to introduce a novel abstract profile that extends the APPOB. This profile should extend the set of compatible executable completions and thus make the implementation of the abstract process more flexible. Considering the travel agency service, the following variations should be allowed:

- The opaque activities describing the preparations of the orders for the hotel service and the airline service do not need to be implemented in *two* distinct activities. Instead, a single activity (maybe embedded into a loop) should be allowed in a compatible executable completion.
- The order in which the reservation requests are sent to the hotel and the airline should be more relaxed. The hotel and the airline do not communicate with each other and therefore cannot realize whether the hotel reservation request is sent *before* or *after* the airline reservation request. Thus, any

```

<process name="travel_agency" ...>
  <partnerLinks>
    <partnerLink name="client" ... />
    <partnerLink name="hotel" ... />
    <partnerLink name="airline" ... />
  </partnerLinks>
  <sequence>
    <receive partnerLink="client"
      operation="travel_request"
      variable="##opaque" />
    <opaqueActivity name="prepare hotel order" />
    <opaqueActivity name="prepare airline order" />
    <invoke partnerLink="hotel"
      operation="hotel_request"
      inputVariable="##opaque" />
    <invoke partnerLink="airline"
      operation="flight_request"
      inputVariable="##opaque" />
    <receive partnerLink="hotel"
      operation="hotel_offer"
      variable="##opaque" />
    <receive partnerLink="airline"
      operation="flight_offer"
      variable="##opaque" />
    <opaqueActivity name="merge offers" />
    <invoke partnerLink="client"
      operation="travel_offer"
      inputVariable="##opaque" />
  </sequence>
</process>

```

Fig. 2. Abstract process describing the observable behavior of a travel agency service.

reordering or even a concurrent execution of the invoke activities sending the requests should be allowed if an asynchronous message binding is assumed.

- Likewise, a reordering or concurrent execution of the receive activities should be allowed, again assuming an asynchronous message binding.

For this and other variations, we will present transformation rules that ensure equivalence of the observable behavior. In Sect. 5, we present their application to the travel agency service.

3 Formalizing Service Conformance

In [6], we proposed an approach for a contract-based composition of services which is based on formal models of services. A formal service model can be generated from a real (BPEL) specification using, for instance, the feature-complete Petri net semantics [7] for BPEL which is available as an implemented translation from BPEL to Petri nets using the tool BPEL2oWFN [3].

On the level of service models, a *strategy* of a given finite state service S is another finite state service S' with compatible interface such that the composition of S with S' forms a deadlock free finite state system. In [4], we show that we can compute a finite representation for the set $Strat(S)$ of all strategies of a given service S . We call this representation an *operating guideline* for S .

as it describes all correct interaction scenarios with S . The computation of an operating guideline is implemented in our tool Fiona [3].⁴

In the approach of [6], a contract is a set of service models (one for each party involved in the contract) such that their composition is deadlock free. Each of these services is called a *public view* of one party's contribution to the overall process. The verification of deadlock freedom is indeed feasible as all public views are available to each participating party.

In contrast to a public view, a private view is the actual implementation of a party's contribution to the overall system. That is, the composition of the private views forms the implementation of the contract. In order to ensure deadlock freedom of the implementation, we proposed a criterion that relates public and private views of a single party. A private view Pr *conforms* to a public view Pu (of the same party) if and only if $Strat(Pu) \subseteq Strat(Pr)$. Intuitively, this means that every strategy of the public view must be a strategy of the private view. If all private views conform to their respective public views, we can show that the implementation of a contract inherits deadlock freedom from the contract itself. The actual value of this approach is that conformance between public and private view can be checked *locally* by each party (using the concept of operating guidelines mentioned above), thus being able not to disclose trade secrets which might be present in a private view.

In our formal model, we assume asynchronous communication (i.e., messages can overtake each other) between the involved parties. Some preliminary considerations on synchronous communication, however, suggest that similar results can be obtained for synchronous, or even a mixture of synchronous and asynchronous communication as well.

In [8], we extended this approach with a set of (syntactic) rules for transforming services such that the conformance relation is established between the input of a transformation and the resulting service. This way, a conforming private view can be systematically derived from a public view.

4 A novel abstract profile for BPEL

Using the formal model introduced above, we are able to formally reason about the behavior of a process. In particular, we are able to define rules that restrict the set of permitted executable completions that are motivated by the *semantics* of a process instead of its *syntax*. These rules are less restrictive than the rules of the APPOB and thereby are suitable to extend BPEL's compatibility relation.

4.1 Equivalence Notion for BPEL Processes

The APPOB in BPEL allows adding new partner links and communicating activities using these new partner links. We introduce a variation of the APPOB

⁴ BPEL2oWFN and Fiona are available under
<http://www.informatik.hu-berlin.de/top/tools4bpe1>.

that preserves the externally observable behavior globally; that is, the set of *all* partner links and interactions across these partner links remain invariant.

In order to do so, we take the existing APPOB and introduce a new profile, the *Abstract Process Profile for Globally Observable Behavior* (APPGOB). The only difference to the existing profile is that we do not allow adding new partner links as part of the executable completion. Therefore, in a sense, we thereby restrict the set of executable completions. Furthermore, in the BPEL specification, the common base is too restrictive in only allowing replacements of opaque entities and insertions of additional executable entities. There exist a number of cases where the reordering of activities as well as the deletion of activities can be tolerated if these transformations do not affect the main intention of the profile. The APPOB (as well as the more restrictive APPGOB) concentrate on interactions across partner links. Other activities like simple assignments do not need to be handled in such a restrictive fashion. For example, several assignment operations may be independent in the sense that reordering them has no effect on the externally observable behavior of the process. Therefore, we introduce a new relation we call “behavioral equivalence”. The purpose of this relation is to extend the set of executable completions to a set of executable processes exposing the same observable behavior. We define our *behavioral equivalence* relation as follows:

Definition 1 (Behavioral equivalence). *Let ep_1 and ep_2 be executable processes. Then ep_1 is behavioral equivalent to ep_2 if and only if ep_1 can be created out of ep_2 by applying zero or more of the following eight transformation rules:*


1. *Looping Existing Activity*
2. *Activity Removal from Sequence*
3. *Activity Removal from Flow*
4. *Activity Reordering*
5. *Invoke-Flow Serialization*
6. *Receive-Flow Serialization*
7. *Invoke and Receive*
8. *Communicating If-branches*

In the following, we introduce the above mentioned rules. We distinguish between the first four rules on the one hand as they consider only non-communicating activities and the remaining rules for communicating activities on the other hand. For each rule we present a textual description and an illustrating example by help of a code snippet. On the left hand side of each example the part of the existing BPEL process is shown while on the right hand side the respective part after applying the transformation is illustrated.

Rules for Non-Communicating Activities We have identified the following four rules for non-communicating activities:

Rule 1: Looping Existing Activity. Given a sequence of activities, we can embed a present non-communicating activity into a finite (while/repeatUntil/forEach) loop.

Example for Rule 1:



```

<sequence>
  <activity1 />
  <activity2 />
  <activity3 />
</sequence>
  
```


```

<sequence>
  <activity1 />
  <while>
    <condition ... />
    <activity2 />
  </while>
  <activity3 />
</sequence>
  
```

Although the common base already allows for inserting activities, it does not consider that a present non-communicating activity can be embedded into a loop.

Rule 2: Activity Removal from Sequence. A non-communicating basic or structured activity can be deleted from a sequence of activities.

Example for Rule 2:



```

<sequence>
  <activity1 />
  <activity2 />
  <activity3 />
</sequence>
  
```


```

<sequence>
  <activity1 />
  <activity3 />
</sequence>
  
```

The common base allows for inserting an activity. That means, we can apply the transformation illustrated in the above example in the other direction as well. However, also removing an activity from a sequence does not change the observable behavior.

Rule 3: Activity Removal from Flow. A non-communicating basic or structured activity can be deleted from a flow.

Example for Rule 3:



```

<flow>
  <activity1 />
  <activity2 />
  <activity3 />
</flow>
  
```

```

<flow>
  <activity1 />
  <activity3 />
</flow>
  
```


Like for the last rule, the common base only considers activity insertion but not the removal of a non-communicating activity from a flow.

Rule 4: Activity Reordering. A sequence of solely non-communicating basic or structured activities can be arbitrarily reordered.

Example for Rule 4:

```

<sequence>
  <activity1 />
  <activity2 />
</sequence>
  
```



```

<sequence>
  <activity2 />
  <activity1 />
</sequence>
  
```

Since we are allowed to remove and to insert non-communicating activities, we can consequently also reorder non-communicating activities that are sequentially ordered. It is important to mention that applying these rules must not violate data-dependencies between activities.


Rules for Communicating Activities Next we introduce four additional rules that change the order of communicating activities.

Rule 5: Invoke-Flow Serialization. A flow of one-way invoke activities can be transformed into a sequence.

Example for Rule 5:

```

<flow>
  <invoke operation="a" />
  <invoke operation="b" />
</flow>
  
```



```

<sequence>
  <invoke operation="b" />
  <invoke operation="a" />
</sequence>
  
```


If n one-way invoke activities are concurrently executed in a flow, then these activities can be executed in any sequential order without changing the observable behavior. Rule 5 reflects the fact that every permutation of the one-way invoke activities is a possible execution sequence due to the concurrency in the flow activity. Informally spoken, applying Rule 5 fixes one of these sequences and hence restricts the behavior of the process.

Rule 6: Receive-Flow Serialization. A flow of receive activities can be transformed into a sequence.

Example for Rule 6:

```

<flow>
  <receive operation="a" />
  <receive operation="b" />
</flow>
  
```



```

<sequence>
  <receive operation="b" />
  <receive operation="a" />
</sequence>
  
```


Rule 6 is the analogous of Rule 5; that is, n receive activities being concurrently executed in a flow can be executed in any sequential order without changing the observable behavior.

Rule 7: Invoke and Receive. A sequence of first a one way invoke activity and then a receive activity can be transformed into a flow.

Example for Rule 7:

```

<sequence>
  <invoke operation="a" />
  <receive operation="b" />
</sequence>
  
```



```

<flow>
  <invoke operation="a" />
  <receive operation="b" />
</flow>
  
```


Applying Rule 7 allows to increase the amount of concurrency in the BPEL process without affecting the observable behavior. In case of synchronous binding, a partner has to be the mirrored process, meaning it consists of a sequence of first a receive activity and then a one-way invoke activity. Thus, it will not be affected by the transformation. Otherwise, in case of asynchronous binding, a partner can also consist of a flow embedding a receive activity and a one-way invoke activity which is also not affected by the rule.

Rule 8: Communicating If-branches If there is a BPEL process P with an if activity, where each branch starts with an invoke activity and P 's partner process has a pick activity in which each branch corresponds to one of the invoke activities in the if activity, then a branch can be removed from the if activity.

Example for Rule 8 (the rule is illustrated on top, below the expected partner is shown):

```

<if>
  <condition ... />
  <sequence>
    <invoke operation="a" />
    ...
  </sequence>
  <elseif>
    <condition ... />
    <sequence>
      <invoke operation="b" />
      ...
    </sequence>
  </elseif>
  <else>
    <sequence>
      <invoke operation="c" />
      ...
    </sequence>
  </else>
</if>
  
```



```

<if>
  <condition ... />
  <sequence>
    <invoke operation="a" />
    ...
  </sequence>
  <else>
    <sequence>
      <invoke operation="c" />
      ...
    </sequence>
  </else>
</if>
  
```

```

<pick>
  <onMessage operation="a"> ... </onMessage>
  <onMessage operation="b"> ... </onMessage>
  <onMessage operation="c"> ... </onMessage>
</pick>
  
```

Rule 8 is a little bit more restricted, since it can only be applied if knowledge about the structure of the partner is available. The idea is that since a partner has to be able to receive all sending messages of the corresponding if activity

in P , it still a partner if the number of branches (and therefore the number of invoke activities) in P will be restricted by removing a branch.

A Conformance Notion for WS-BPEL Considering executable processes that are behavioral equivalent to an executable process belonging to the set of all executable completions for a given abstract process, we can construct the transitive relationship between all equivalent executable processes and the abstract process.

Definition 2 (Conformance). *Let ap be an abstract process and let ep_1 be an executable process, let further EC_{GOB} denote the set of executable completions of ap allowed by the APPGOB. Then ep_1 conforms to ap if and only if there exists an executable process $ep_2 \in EC_{GOB}$ such that ep_1 and ep_2 are behavioral equivalent.*

Each executable process in the set EC_{GOB} of executable completions (cf. Fig. 1) that *conforms* to an abstract process exposes the same externally observable behavior. Figure 3 illustrates the relationship between an abstract process ap , an executable process ep_1 that conforms to ap and an executable process ep_2 that is behavioral equivalent to ep_1 as it is defined in Def. 2. Therefore, our proposed conformance notion extends the notion of compatibility presented in the BPEL specification, because it allows to relate executable processes ep_1 with an abstract process ap if ep_1 and ep_2 are behavioral equivalent. That way much more executable processes can be related to an abstract process which can be derived by applying the rules we have presented.

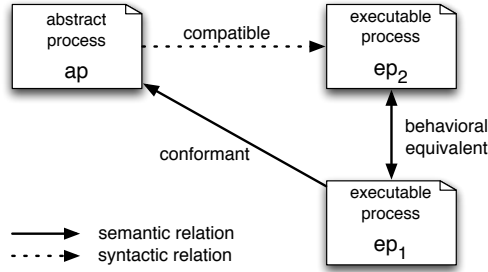


Fig. 3. The relationships between compatibility, behavioral equivalence, and conformance.

4.2 Relaxed Equivalence Relation for Asynchronous Bindings

The BPEL specification makes no assumptions about protocols, bindings, and quality of service attributes of interactions. So far, all presented transformation


rules are valid for both synchronous and asynchronous binding. However, if we would assume asynchronous bindings, we can relax the *behavioral equivalence* relationship even further by introducing three additional transformation rules. We therefore generalize the relation of behavior-equivalence to relaxed behavioral equivalence.

Definition 3 (Relaxed behavioral equivalence). *Let ep_1 and ep_2 be executable processes. Then ep_1 is relaxed behavioral equivalent to ep_2 if and only if ep_1 is behavioral equivalent to ep_2 , or ep_1 can be created out of ep_2 by applying zero or more of the following two additional transformation rules:*

9. *Invoke-Sequence Reordering*
10. *Receive-Sequence Reordering*
11. *Invoke and Receive*

Rule 9: Invoke-Sequence Reordering. A sequence of one-way invoke activities can be arbitrarily reordered or it can be transformed into a flow.


Example for Rule 9:

<pre><sequence> <invoke operation="a" /> <invoke operation="b" /> </sequence></pre>		<pre><sequence> <invoke operation="b" /> <invoke operation="a" /> </sequence></pre>
---	--	---

In case of asynchronous binding it is possible that if a process sends first a message *a* and then a message *b*, then its partner process may receive *b* before *a*. This is caused by the fact that messages can overtake each other on a message channel. Since a sequence of *n* send messages can reach the partner in an any order, we can arbitrarily reorder a sequence of one-way invoke activities or even embed these activities into a flow.

Rule 10: Receive-Sequence Reordering. A sequence of receive activities can be arbitrarily reordered or it can be transformed into a flow.

Example for Rule 10:

<pre><sequence> <receive operation="a" /> <receive operation="b" /> </sequence></pre>		<pre><sequence> <receive operation="b" /> <receive operation="a" /> </sequence></pre>
---	---	---


For the same arguments as presented for Rule 9, we can arbitrarily reorder a sequence of receive activities and or even embed these activities into a flow by applying Rule 10.

Rule 11: Invoke and Receive. A flow that contains a one-way invoke activity and a receive activity can be transformed into a sequence of *firstly* the invoke and *then* the receive activity.

Example for Rule 11:

```

<flow>
  <invoke operation="a" />
  <receive operation="b" />
</flow>
  
```



```

<sequence>
  <invoke operation="a" />
  <receive operation="b" />
</sequence>
  
```

Rule 11 describes in fact the opposite direction of Rule 7. It is only applicable in case of asynchronous bindings: Assume a process similar to the left hand side of the example. Further assume a mirrored version of this process as a partner. Applying Rule 11 to the process in case of an synchronous binding could lead to a deadlocking situation in the case where the message on *b* sent by the partner arrives before the process has sent the message on *a* to the partner.

4.3 Disallowed Transformation Rules


Based on the results presented above, the notion of equivalent behavior can be significantly extended beyond BPEL's APPOB. However, even for our extended conformance notion, there exist limitations, as shown by the following three transformation rules which are explicitly disallowed.

Anti-rule 1. A sequence of first a one-way invoke and then a receive activity **MUST NOT** be reordered, or vice versa.

Example for Anti-rule 1:

```

<sequence>
  <invoke operation="a" />
  <receive operation="b" />
</sequence>
  
```



```

<sequence>
  <receive operation="b" />
  <invoke operation="a" />
</sequence>
  
```


Beside reordering, also a concurrent execution is disallowed:

Anti-rule 2. A sequence of first a receive activity and then a one way invoke activity **MUST NOT** be transformed into a flow, or vice versa.

Example for Anti-rule 2:

```

<sequence>
  <receive operation="a" />
  <invoke operation="b" />
</sequence>
  
```



```

<flow>
  <receive operation="a" />
  <invoke operation="b" />
</flow>
  
```

Finally, the addition and usage of new partner links is not permitted. This anti-rule differs from the original APPOB where this addition is explicitly allowed.

Anti-rule 3. New partner links or communicating activities **MUST NOT** be added.

We discuss the motivation and the validity of the Anti-rules in the next subsection.

4.4 Discussion

So far, we just listed a set of rules and left the task of verification of their correctness to the reader's intuition. Correctness can, however, as well be formally verified. To this end, it is possible to map the given rules into the formalism of Petri nets using, for instance, the semantics of [7]. It turns out, that the resulting Petri net transformation rules correspond to those that have been proven formally correct in [8].

To justify the Anti-rules we show that applying these rules to a process P would exclude a partner of P in the resulting transformed process P' . Consider again Anti-rule 1. The process depicted in Fig. 4(a) is a partner for the left pattern in the example of Anti-rule 1, but not for the right pattern. Furthermore, Fig. 4(b) depicts a partner for the right pattern which is no valid partner for the left pattern. Similar for Anti-rule 2, Fig. 4(c) is a partner of the left hand side of the example in Anti-rule 2 but not for the right hand side. In contrast, Fig. 4(d) depicts a valid partner for the right hand side but not for the left hand side of Anti-rule 2.

<pre> <sequence> <receive operation="a" /> <invoke operation="b" /> </sequence> </pre> <p style="text-align: center;">(a)</p>	<pre> <sequence> <invoke operation="b" /> <receive operation="a"/> </sequence> </pre> <p style="text-align: center;">(b)</p>	<pre> <sequence> <receive operation="b" /> <invoke operation="a" /> </sequence> </pre> <p style="text-align: center;">(c)</p>
<pre> <pick> <onAlarm> <sequence> <invoke operation="b" /> <receive operation="a" /> </sequence> </onAlarm> <onMessage operation="a" /> <invoke operation="c" /> </onMessage> </pick> </pre> <p style="text-align: center;">(d)</p>		

Fig. 4. Counterexamples justifying the correctness of Anti-rules 1 and 2.

Anti-rule 3 excludes the addition of new partner links and new communicating activities. While this is permitted in the original APPOB and does not affect the observable behavior from one partner's point of view, it would change the global observable behavior by introducing “unintended” behavior. As an example, consider the process depicted in Fig. 5.

```

<process name="example" ...>
  <partnerLinks>
    <partnerLink name="a" ... />
    <partnerLink name="b" ... />
  </partnerLinks>
  <sequence>
    <receive partnerLink="a" operation="a1" ... />
    <if>
      <condition> ... </condition>
      <sequence>
        <empty />
        <receive partnerLink="b" operation="b1" ... />
      </sequence>
      <else>
        <sequence>
          <empty />
          <receive partnerLink="b" operation="b2" ... />
        </sequence>
      </else>
    </if>
  </sequence>
</process>

```

Fig. 5. Counterexample justifying the correctness of Anti-rule 3.

The original APPOB would allow the addition of a partner link and the two receive activities (the bold lines of Fig. 5). Though this addition would not affect the partner communicating via partner link *a*, the addition would introduce unintended behavior: A partner communicating via partner link *b* would have to *guess* how the condition of the if activity was evaluated to decide whether to send a message using operation *b1* or *b2*. Therefore, the process could either deadlock (in case the wrong operation was used) or would complete with a redundant pending message (in case both operations were used). Both cases are certainly not desirable, though not excluded by the APPOB.

Finally, it is worthwhile mentioning that the presented (syntactic) rules are correct (i. e., the application of the rules preserves conformance), but not complete (i. e., there exist conforming processes that cannot be derived by applying the presented transformation rules). As an alternative to these rules, however, we can check (relaxed) behavioral equivalence between two BPEL processes a-posteriori with our tools BPEL2oWFN/Fiona.

5 Example revisited

The novel abstract profile, APPGOB, now allows the modifications motivated in Sect. 2.2. In particular, the reordering of the communicating activities can now be achieved by applying transformation rules that guarantee observable behavioral equivalence. Figure 6 illustrates the applied transformation rules and their effects. We assume an asynchronous binding for the travel agency service.

Firstly, the opaque activities that organize the message sent from or to the agency are replaced by assign activities. Then, those activities that prepare the orders for the hotel and airline reservation system are removed (Rule 2). Instead, a new opaque activity is added and embedded into a while loop (Rule 1). This

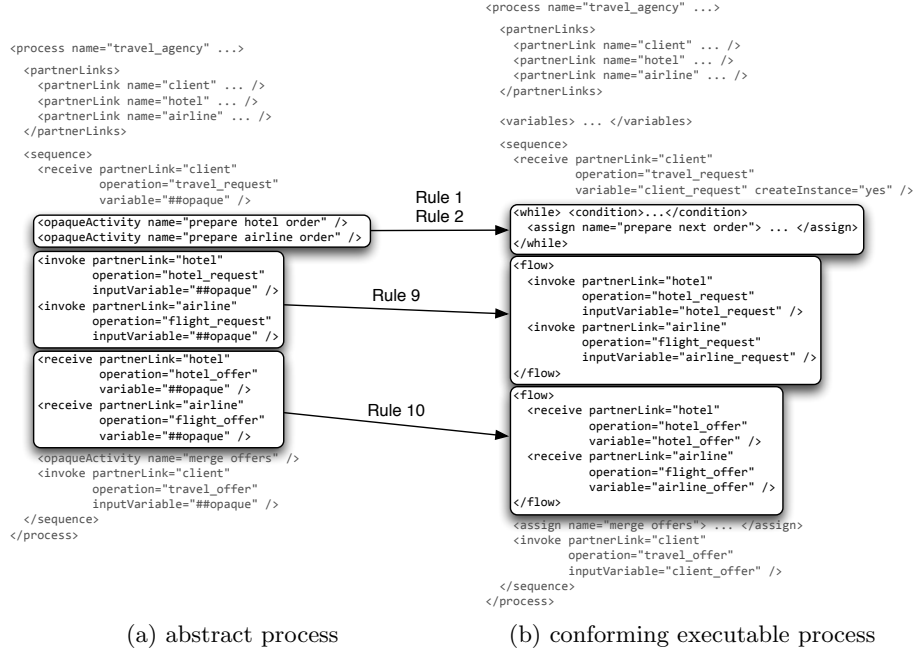


Fig. 6. Application of transformation rules to the abstract travel agency process.

loop allows more flexibility and an easy integration of additional parties such as, for example, a car rental agency. Then, the invoke activities sending the requests to the hotel and the airline are embedded into a flow activity (Rule 9). Similarly, the responses of the hotel and the airline services are received concurrently (Rule 10).

By Definition 3, the derived executable service (cf. Fig. 6(b)) is relaxed behavioral equivalent to the executable process in which only the opaque activities are replaced by assign activities. Furthermore, the executable process derived by applying the transformation rules conforms to the abstract process of the travel agency in Fig. 6(a). Still, this process would not be considered compatible by the original APPOB.

6 Related Work

There are many other papers dealing with conformance notions. Basten and van der Aalst present in [9] the notion of projection inheritance for Petri nets. Two Petri nets are related under projection inheritance if they have the same observable behavior. In [8] we have proven that our notion of conformance is a generalization of projection inheritance, meaning projection inheritance implies conformance. Rules 1–4 for non-communicating activities presented in Sect. 4.1 preserve projection inheritance. All other rules influence the communicating behavior and therefore do not preserve projection inheritance.

Several authors propose conformance notions using process calculi. Fournet et al. [10] present stuck-free conformance, a refinement relation between two CCS processes of asynchronous message passing software components. Stuck-freedom formalizes like our notion of deadlock freedom the absence of deadlocks in the system. Bravetti and Zavattaro [11] propose a conformance notion that guarantees the absence of deadlocks and livelocks in cyclic systems. In [12], their correctness criterion is enhanced by ensuring whenever a message can be sent, the other service is ready to receive this message. Systems that behave this way are called strong compliant. Strong compliance can therefore be used as a correctness criterion for BPEL choreographies in case synchronous bindings are used. Castagna et al. [13] formalize (similar to our notion of deadlock freedom) the absence of deadlocks in finite-state systems. This notion is as in [12] called strong compliance. In addition they propose the notion of weak compliance. Two services are weak compliant if they can be made strong compliant by applying message filters on these services to hide specific actions. A more expressive calculi which supports name passing is used by Carbone et al. [14]. In contrast to our notion of conformance, none of these conformance relations allows the reordering of messages.

Busi et al. [15] present a notion of conformance between a choreography language based on WS-CDL and an orchestration language based on abstract BPEL. In fact, this conformance notion is branching bisimulation. Conformance can be used to check if the implementation (i.e., the orchestrated system) behaves accordingly the conversation rules of the choreography. Bonchi et al. [16] model the behavior of services using a special kind of Petri nets, Consume-Produce-Read Nets. For their model they present saturated bisimulation as conformance relation. Both branching bisimulation and saturated bisimulation are too restrictive to allow reordering of messages.

The concept of contract conformance is also related to deciding when a service can be substituted by another service. Most of this work, however, is restricted to synchronous communication [17–19] whereas our service model considers asynchronous message passing. Benatallah et al. [19] present four notions of substitutability. In this paper, we cover two of them: equivalence and subsumption. Equivalence in our notion means that both services have the same set of strategies and subsumption means the inclusion of the set of strategies.

Lohmann et al. translate in [20] choreographies specified in the choreography language BPEL4Chor [21] into a Petri net model. This model is then checked for deadlocks using the model checker LoLA [22].

To summarize, most of the work uses a synchronous communication model whereas our model is based on asynchronous message passing thus allowing to identify rules as shown in Sect. 4.3. Furthermore, except the work on projection inheritance [9] there are to the best of our knowledge no papers about rules to derive from a service S a service S' that conforms to S .

7 Conclusion

In this paper, we have presented a more liberal approach to decide compatibility between an abstract and an executable BPEL process. Therefore, we defined a novel profile for BPEL. This novel profile enhances the existing Abstract Process Profile for Observable Behavior (APPOB) by introducing a notion for behavioral equivalence on the one hand, and restricting it by defining explicit anti-rules on the other hand. We have shown that with our novel profile more executable processes can be considered conformant to an abstract process without the loss of general applicability.

Based on our notion of behavioral equivalence we have identified and proven a set of transformation rules. Given an abstract process ap and an executable process ep that is compatible to ap with respect to the APPOB, these rules can be applied to derive an executable process ep' that conforms to ap from ep . The set of presented rules is twofold. We have presented transformation rules being applicable both for synchronous and asynchronous bindings and in addition, we have introduced additional (more relaxed) transformation rules in case that an asynchronous binding is used.

Acknowledgments Niels Lohmann is funded by the German Federal Ministry of Education and Research (project Tools4BPEL, project number 01ISE08). Christian Stahl is funded by the DFG project “Substitutability of Services” (RE 834/16-1).

References

1. Alves, A., Arkin, A., Askary, S., Barreto, C., Bloch, B., Curbera, F., Ford, M., Golland, Y., Guízar, A., Kartha, N., Liu, C.K., Khalaf, R., König, D., Marin, M., Mehta, V., Thatte, S., Rijn, D.v.d., Yendluri, P., Yiu, A.: Web Services Business Process Execution Language Version 2.0. OASIS Standard, 11 April 2007, OASIS (2007)
2. Moser, S., Martens, A., Häbich, M., Müller, J.: A hybrid approach for generating compatible ws-bpel partner processes. In Dustdar, S., Fiadeiro, J.L., Sheth, A.P., eds.: Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings. Volume 4102 of Lecture Notes in Computer Science., Springer (2006) 458–464
3. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: 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., Springer-Verlag (2006) 17–32
4. Lohmann, N., Massuthe, P., Wolf, K.: Operating guidelines for finite-state services. In Kleijn, J., Yakovlev, A., eds.: 28th International Conference on Applications and Theory of Petri Nets and Other Models of Concurrency, ICATPN 2007, Siedlce, Poland, June 25-29, 2007, Proceedings. Volume 4546 of Lecture Notes in Computer Science., Springer-Verlag (2007) 321–341

5. Ambühler, T.: UML 2.0 profile for WS-BPEL with mapping to WS-BPEL. Master's thesis, Universität Stuttgart, Institut für Architektur von Anwendungssystemen, Stuttgart, Germany (2005)
6. Aalst, W.M.P.v.d., Massuthe, P., Stahl, C., Wolf, K.: Multiparty contracts: Agreeing and implementing interorganizational processes. *Informatik-Berichte* 213, Humboldt-Universität zu Berlin, Berlin, Germany (2007)
7. Lohmann, N.: A feature-complete Petri net semantics for WS-BPEL 2.0. In Dumas, M., Heckel, R., eds.: *Web Services and Formal Methods, Forth International Workshop, WS-FM 2007 Brisbane, Australia, September 28-29, 2007, Proceedings. Lecture Notes in Computer Science*, Springer-Verlag (2007)
8. Aalst, W.M.P.v.d., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: From public views to private views – correctness-by-design for services. In Dumas, M., Heckel, R., eds.: *Web Services and Formal Methods, Forth International Workshop, WS-FM 2007 Brisbane, Australia, September 28-29, 2007, Proceedings. Lecture Notes in Computer Science*, Springer-Verlag (2007)
9. Basten, T., Aalst, W.M.P.v.d.: Inheritance of behavior. *J. Log. Algebr. Program.* **47**(2) (2001) 47–145
10. Fournet, C., Hoare, C.A.R., Rajamani, S.K., Rehof, J.: Stuck-free conformance. In Alur, R., Peled, D., eds.: *Computer Aided Verification, 16th International Conference, CAV 2004, Boston, MA, USA, July 13-17, 2004, Proceedings. Volume 3114 of Lecture Notes in Computer Science.*, Springer (2004) 242–254
11. Bravetti, M., Zavattaro, G.: Contract based multi-party service composition. In Arbab, F., Sirjani, M., eds.: *International Symposium on Fundamentals of Software Engineering, International Symposium, FSEN 2007, Tehran, Iran, April 17-19, 2007, Proceedings. Volume 4767 of Lecture Notes in Computer Science.*, Springer (2007) 207–222
12. Bravetti, M., Zavattaro, G.: A theory for strong service compliance. In Murphy, A.L., Vitek, J., eds.: *Coordination Models and Languages, 9th International Conference, COORDINATION 2007, Paphos, Cyprus, June 6-8, 2007, Proceedings. Volume 4467 of Lecture Notes in Computer Science.*, Springer (2007) 96–112
13. Castagna, G., Gesbert, N., Padovani, L.: A Theory of Contracts for Web Services. In: accepted at POPL 2008. (2008)
14. Carbone, M., Honda, K., Yoshida, N.: A calculus of global interaction based on session types. *Electr. Notes Theor. Comput. Sci.* **171**(3) (2007) 127–151
15. Busi, N., Gorrieri, R., Guidi, C., Lucchi, R., Zavattaro, G.: Choreography and orchestration: A synergic approach for system design. In Benatallah, B., Casati, F., Traverso, P., eds.: *Service-Oriented Computing - ICSOC 2005, Third International Conference, Amsterdam, The Netherlands, December 12-15, 2005, Proceedings. Volume 3826 of Lecture Notes in Computer Science.*, Springer (2005) 228–240
16. Bonchi, F., Brogi, A., Corfini, S., Gadducci, F.: A Behavioural Congruence for Web Services. In Arbab, F., Sirjani, M., eds.: *International Symposium on Fundamentals of Software Engineering, International Symposium, FSEN 2007, Tehran, Iran, April 17-19, 2007, Proceedings. Volume 4767 of Lecture Notes in Computer Science.*, Springer (2007) 240–256
17. Bordeaux, L., Salaün, G., Berardi, D., Mecella, M.: When are two Web services compatible? In Shan, M.C., Dayal, U., Hsu, M., eds.: *Technologies for E-Services, 5th International Workshop, TES 2004, Toronto, Canada, August 29-30, 2004, Revised Selected Papers. Volume 3324 of Lecture Notes in Computer Science.*, Springer (2004) 15–28

18. Beyer, D., Chakrabarti, A., Henzinger, T.A.: Web service interfaces. In Ellis, A., Hagino, T., eds.: Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005, ACM (2005) 148–159
19. Benatallah, B., Casati, F., Toumani, F.: Representing, analysing and managing Web service protocols. *Data Knowl. Eng.* **58**(3) (2006) 327–357
20. Lohmann, N., Kopp, O., Leymann, F., Reisig, W.: Analyzing BPEL4Chor: Verification and participant synthesis. In Dumas, M., Heckel, R., eds.: *Web Services and Formal Methods, Forth International Workshop, WS-FM 2007 Brisbane, Australia, September 28-29, 2007, Proceedings. Lecture Notes in Computer Science*, Springer-Verlag (2007)
21. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4Chor: Extending BPEL for Modeling Choreographies. In: 2007 IEEE International Conference on Web Services (ICWS 2007), July 9-13, 2007, Salt Lake City, Utah, USA, IEEE Computer Society (2007) 296–303
22. Schmidt, K.: LoLA: A Low Level Analyser. In Nielsen, M., Simpson, D., eds.: *Application and Theory of Petri Nets, 21st International Conference (ICATPN 2000)*. Number 1825 in *Lecture Notes in Computer Science*, Springer-Verlag (2000) 465–474