A Parametric Communication Model for the Verification of BPEL4WS Compositions [⋆]

Raman Kazhamiakin and Marco Pistore

DIT, University of Trento
via Sommarive 14, 38050, Trento, Italy
{raman,pistore}@dit.unitn.it

Abstract. In this paper we describe an approach for the verification of Web service compositions defined by a set of BPEL4WS processes. The key aspect of such a verification task is the model adopted for representing the communications among the services participating to the composition. Indeed, these communications are asynchronous and buffered in the existing execution frameworks, while most verification approaches adopt a synchronous communication model for efficiency reasons. In our approach, we model the asynchronous nature of Web service interactions without introducing buffers, by allowing a reordering of the messages exchanged during these interactions. This way, we can provide an accurate model of a wider class of service composition scenarios, while preserving an efficient performance in verification.

1 Introduction

Web services provide the basis for the development and execution of business processes that are distributed over the network and available via standard interfaces and protocols [9]. Service composition [10] is one of the most promising ideas underlying Web services: new functionalities can be defined and implemented by combining and interacting with pre-existing services. Different standards and languages have been proposed to develop Web service compositions. BPEL4WS (Business Process Execution Language for Web Services, BPEL for short) [3] is one of the emerging standards for describing a key aspect for the composition of Web services: the behavior of the service. It provides a core of process description concepts that allow for the definition of business processes interactions. This core of concepts is used both for defining the internal *business processes* of a participant to a business interaction and for describing and publishing the external *business protocol* that defines the interaction behavior of a participant without revealing its internal behavior.

BPEL opens up the possibility of applying a range of formal techniques to the verification of the behavior of Web services, and different approaches have been

[⋆] This work is partially funded by the MIUR-FIRB project RBNE0195K5, "Knowledge Level Automated Software Engineering", and by the MIUR-PRIN 2004 project "Advanced Artificial Intelligence Systems for Web Services".

defined for verifying BPEL [7, 11, 13, 12, 8, 15]. We are interested in particular in those techniques that are applied to the verification of BPEL compositions: in this case, we have to verify the behaviors generated by the interactions of a set of BPEL processes, each specifying the workflow and the protocol of one of the services participating to the composition.

A key aspect for this kind of verification is the model adopted for representing the communications among the Web services. Indeed, the actual mechanism implemented in the existing BPEL execution engines is both very complex and implementation dependent. More precisely, BPEL processes exchange messages in an asynchronous way; incoming messages go through different layers of software, and hence through multiple queues, before they are actually consumed in the BPEL activity; and overpasses are possible among the exchanged messages.

On the other hand, most of the approaches proposed for a formal verification of BPEL compositions are based on a synchronous model of communications, which does not require message queues and hence allows for a better performance in verification. This synchronous mechanism relies on some strong hypotheses on the interactions allowed in the composition: at a given moment in time, only one of the components can emit a message, and the receiver of that message is ready to accept it (see e.g., [8]).

In our experience, these hypotheses are not satisfied by many Web service composition scenarios of practical relevance, where critical runs can happen among messages emitted by different Web services. This is the case, for instance, when a Web service can receive inputs concurrently from two different sources, or when a service which is executing a time consuming task can receive a cancellation message before the task is completed.

Our goal is to provide extended composition mechanisms, where the hypotheses on synchronous communications are weakened, but an explicit introduction of message queues is still not required. This way, an accurate modelling is possible for a wider class of service composition scenarios, while an efficient performance is still possible.

In this paper, we propose a parametric model of composition, which is based on synchronous communications, but which allows for a reordering of the exchanged messages in order to model critical runs and message overpasses that may occur in the execution of BPEL processes. More precisely, we define three variants of this mechanism, depending on the degree of reordering allowed in the messages. The first variant, where no reordering is possible, corresponds to the synchronous model of [8]. The second variant permits to reorder only messages sent or received by different partners, that is, it takes into account that a difference between the order of emission and the order of reception of the messages due to the distributed nature of Web services. The third variant, finally, allows for reordering messages also between the same two partners, thus considering message overpasses that can occur in the message queues of the BPEL engines.

For each of the three composition models, we define a validity check, that determines whether the model is adequate for a given composition scenario. Moreover, we define a composition and verification algorithm that is correct and complete for those scenarios that pass the validity check. We have implemented the proposed approach and we report our preliminary experiments in its application to a case study based on a Virtual Travel Agency domain.

The paper is structured as follows. In Sect. 2 we introduce several instances of the case study that motivate the necessity to consider different variants of communication mechanism. Section 3 explains how BPEL processes can be translated into state transition systems. We give the formal definition of the extended composition and the notion of composition validity in Sect. 4, and describe different models basing on these notions in Sect. 5. Section 6 explains the architecture of the described analysis framework and reports the results of its experimental evaluation. Conclusions and future work are presented in Sect. 7.

2 Modelling BPEL Compositions

In order to illustrate the problem of modelling BPEL compositions we consider several variants of the Virtual Travel Agency domain. The goal of the Virtual Travel Agency is to provide a combined flight and hotel booking service by integrating separate, independent existing services: a Flight booking service, and a Hotel booking service. Thus, the composition describes the interactions of four partners: User, Virtual Travel Agency (VTA), Hotel and Flight services (see Fig. 1.a). We model the composition using BPEL specifications that describe the workflows and the interactions of the four partners.

Example 1: Tickets Reservation Scenario. The case study describes the behavior exposed by VTA that allows the user to book a flight to the specified place and reserve a room in the hotel at that place for a given period of time. Provided a reservation offer, the user can accept or reject it, sending a corresponding message to the VTA service (Fig. 1.b).

The Flight booking service becomes active upon a request for a given location (e.g., Paris) and a given period of time (e.g., August, 15-20). In the case the booking is not possible, this is signaled to the requestor, and the protocol terminates. Otherwise, the requestor is notified with an offer information and the protocol stops waiting for either a positive or negative acknowledgment. In case of positive answer the flight is successfully booked and the reservation ticket is sent, otherwise the interaction terminates with failure. Figure 1.c represents the protocol provided by the Flight booking service. The protocol of the Hotel service is similar.

The behavior of the VTA is as follows. Having received a reservation request from the user, VTA interacts with Flight and Hotel services to obtain ticket offers and expects either a negative answer if this is not possible (in which case the user is notified and the protocol terminates failing), or provides the user with an offer indicating hotel, flights and cost of the trip. After that the user may either accept or refuse the offer, and in the first case VTA provides the user with the tickets obtained from Hotel and Flight. The diagram corresponding to the BPEL protocol of VTA is represented in Fig. 1.d.

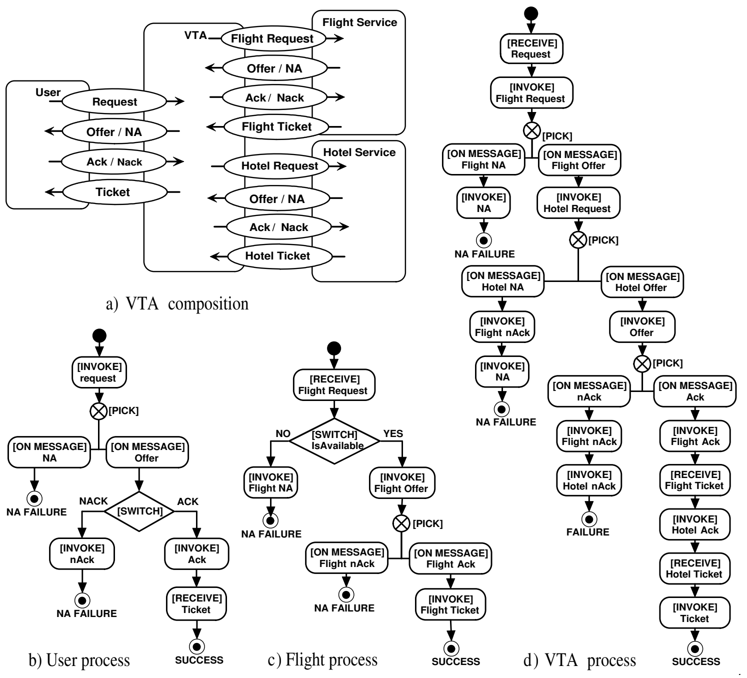


Fig. 1. Composition participants

The scenario exhibits an important property that allows for a very simple communication mechanism. At any moment of time only one message can be emitted by one of the partners. Moreover, such a message is acceptable by the corresponding receiver. Using the terminology of [8], the composition model satisfies the *synchronous compatibility*, *autonomy* and *lossless composition* properties. As a consequence, a synchronous communication model can be used to define the composition without loosing completeness of behaviors.

Example 2: Reservation with Cancellation. Unfortunately, the simplified communication model of the previous example is not applicable to all kinds of interactions. An indicative example is the business process with event handlers. Let us consider an extension of the above case study, such that the User, after having acknowledged the provided offer, can decide to cancel booking operation. In this case the User sends a `Cancel` message to the VTA process and waits for an outcome of the cancellation. The cancellation is forwarded to the Flight process (and similarly to the Hotel process, we omit this for the sake of simplicity). The latter waits a certain time for a cancellation message and if it is received, sends the notification about successful cancellation. Or, if time runs out, sends a ticket to the VTA thus forcing the failure of cancellation; then it consumes the cancellation sent by the VTA and ignores it. The excerpts of the corresponding process specifications are represented in Fig. 2.

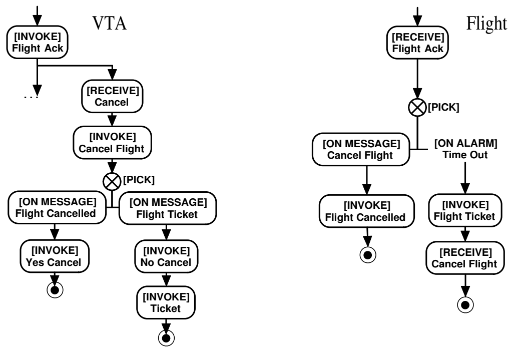


Fig. 2. VTA and Flight processes parts for the cancellation management

The verification under the synchronous communication model is not able to manage this example correctly and reports a deadlock. Indeed, if the Flight service fails to wait for a cancellation, the `onAlarm` activity is fired and it then tries to send a ticket to the VTA process. Meanwhile, the latter receives a cancellation message from User and then tries to send the cancellation to the Flight service. Therefore both services will try to send messages to each other and the composition is in a deadlock, since this is not acceptable by the synchronous semantics.

This deadlock is not real, in the sense it does not occur in real BPEL engines; since the Web services communications are asynchronous, and the message emission is not blocking, both processes will emit messages to each other. Both messages will be consumed then and the composition terminates correctly.

The problem we are facing here is that the synchronous model is too strict. The message delivery and processing may require a certain time, thus leading to situations where concurrent message emissions take place. These situations, however, are not allowed in the synchronous communication model. In order to verify correctly the considered example, a relaxed model is needed that allows to consider these concurrent message emissions.

Example 3: Extended Cancellation Scenario Let us consider a further modification of the case study. Now the fact that the cancellation is not possible is signalled with the special messages: `NoCancel` for the User and `NoFlightCancel` for the VTA process. Having sent the cancellation to the Flight service, the VTA waits for the message indicating that the cancellation is possible or not. In the latter case it waits for the ticket and sends a ticket to the User. The Flight service on the other side behaves as before with the only difference that, after emitting the ticket and receiving the cancellation, it sends a notification about cancellation rejection (i.e. `NoFlightCancel` message). The corresponding diagrams are represented in Fig. 3.

Even if one verifies the example allowing for concurrent message emission the following incorrect scenario will result. The Flight service sends a ticket and

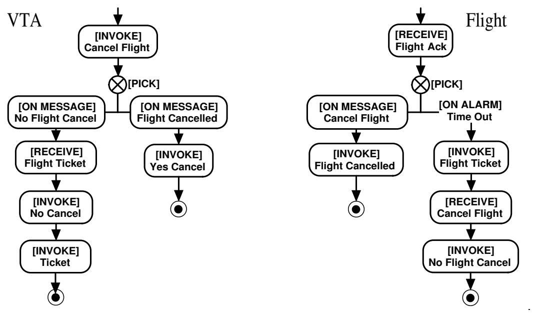


Fig. 3. VTA and Flight processes parts for the complex cancellation management

waits for a cancellation, the VTA process sends a cancellation, the Flight service in turn rejects a cancellation and finishes. The VTA has received a ticket and then a cancellation rejection, but it is not able to process the messages in this order. Only if the execution of processes in the run-time environment allows for reordering of messages (which is the case for existing implementations) the deadlock disappears, since the cancellation rejection can be processed before the ticket message.

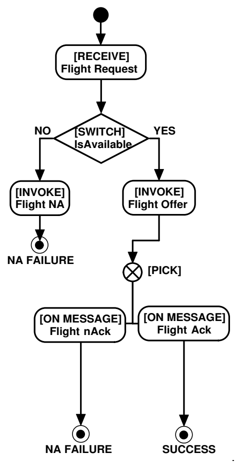
This example shows a necessity not only to consider systems which do not follow the synchronous communication semantics, but also to accept less restrictive models where message reordering is allowed.

This chain can be further prolonged, leading to more complex communication models. One can think of lossy channels, complex ordering conditions, complex queue models etc. Notice, however, that each model requires additional assumptions on the environment where the analyzed system is supposed to execute. These conditions are not always possible to be enforced, or verified. Moreover, the complexity of the verification problem being applied to a certain model significantly varies up to undecidability.

In the following we will introduce the generalized model of the composition, suitable for the analysis of certain classes of communication models.

3 BPEL Processes as State Transition Systems

BPEL provides an operational description of the (stateful) behavior of Web services on top of the service interfaces defined in their WSDL specifications. An abstract BPEL description identifies the partners of a service, its internal variables, and the operations that are triggered upon the invocation of the service by some of the partners. Operations include assigning variables, invoking other services and receiving responses, forking parallel threads of execution, and nondeterministically picking one amongst different courses of actions. Standard imperative constructs such as if-then-else, case choices, and loops, are also supported.



```
PROCESS
  Flight;
TYPE
  Time; Location; Flight;
STATE
  {IN_FlightRequest, SWITCH_IsAvailable, OUT_FlightNA,
   NAFailure, OUT_Offer, PICK_ACK, SUCCESS};
INPUT
  fRequest (t : Time, l : Location); fAck;  fNAck;
OUTPUT
  fOffer (t : Time, f : Flight); fNA;
VAR
  fRequest_location : Location; fRequest_time : Time;
  fOffer_flight     : Flight;   fOffer_time   : Time;
INIT
  state = IN_FlightRequest;
  fRequest_location = fRequest_time = fOffer_flight =
  fOffer_time = UNDEF;
TRANS
  IN_FlightRequest - [IN fRequest(t,l)] -> SWITCH_IsAvailable,
                                           fRequest_time = t,
                                           fRequest_location = l;
  SWITCH_IsAvailable - [TAU] -> OUT_FlightNA;
  SWITCH_IsAvailable - [TAU] -> OUT_Offer;
  OUT_FlightNA - [OUT FligtNA] -> NAFailure;
  OUT_Offer - [OUT fOffer(fOffer_time, fOffer_flight)] -> PICK_ACK;
  PICK_ACK - [IN fAck] -> SUCCESS;
  PICK_ACK - [IN fNAck] -> NAFailure;
```

Fig. 4. The Flight BPEL process and the corresponding STS.

We encode BPEL processes as *state transition systems* which describe dynamic systems that can be in one of their possible *states* (some of which are marked as *initial states*) and can evolve to new states as a result of performing some *actions*. Following the standard approach in process algebras, actions are distinguished in *input actions*, which represent the reception of messages, *output actions*, which represent messages sent to external services, and a special action τ, called *internal action*. The action τ is used to represent internal evolutions that are not visible to external services, i.e., the fact that the state of the system can evolve without producing any output, and independently from the reception of inputs. A *transition relation* describes how the state can evolve on the basis of inputs, outputs, or of the internal action τ.

Definition 1 (State transition system). *A* state transition system Σ *is a tuple* $\langle \mathcal{S}, \mathcal{S}_0, \mathcal{I}, \mathcal{O}, \mathcal{R} \rangle$ *where:*

– $\mathcal{S}$ *is the finite set of states and* $\mathcal{S}_0 \subseteq \mathcal{S}$ *is the set of initial states;*
– $\mathcal{I}$ *is a finite set of input actions and* $\mathcal{O}$ *is a finite set of output actions;*
– $\mathcal{R} \subseteq \mathcal{S} \times (\mathcal{I} \cup \mathcal{O} \cup \{\tau\}) \times \mathcal{S}$ *is the transition relation.*

Figure 4 shows the abstract BPEL process of the Flight service and the corresponding state transition system. The set of states $\mathcal{S}$ models the steps of the evolution of the process and the values of its variables. The special internal variable `state` tracks the information about the current execution step. The other variables (e.g., `fOffer_flight`, `fOffer_time`) correspond to those used by the process to store significant information. In the initial states $\mathcal{S}_0$ all the variables are undefined but `state` that is set to `IN_FlightRequest`.

The evolution of the process is modeled through a set of possible transitions. Each transition defines its applicability conditions on the source state, its firing action, and the destination state. For instance, "`SWITCH_IsAvailable - [TAU] -> OUT_FlightNA`" states that an action τ can be executed in state `SWITCH_IsAvailable` and leads to the state `OUT_FlightNA`.

According to the formal model, we distinguish among three different kinds of actions. The input actions $\mathcal{I}$ model all the incoming requests to the process and the information they bring (i.e., `fRequest` is used for the receiving of the initial request, while `fAck` models the confirmation of the order and `fNAck` its cancellation). The output actions $\mathcal{O}$ represent the outgoing messages (i.e., `FlightNA` is used when there are no tickets for the required date and location, while `fOffer` is used to bid the particular flight for the request). The action τ is used to model internal evolutions of the process, as for instance assignments and decision making (e.g., when the Flight process is in the state `SWITCH_IsAvailable` and performs internal activities to decide whether there are tickets available).

We remark that the definition of the state transition system provided in Fig. 4 is parametric w.r.t. the types `Time`, `Location`, and `Flight` used in the messages. In order to obtain a concrete state transition system, finite ranges have to be assigned to these types.

4 Extended Composition Model

A parallel product with synchronous communications is widely used as a composition model for Web services [7]. As shown in Sect. 2, this model is however not adequate for the description of scenarios where more complicated interactions are essential. We now define an extended composition model that is applicable to those scenarios. We start with some preliminary definitions.

Let $\Sigma = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{I}, \mathcal{O}, \mathcal{R} \rangle$ be an STS. Let $s = s_0, \alpha_0, s_1, \alpha_1, \ldots, \alpha_{n-1}, s_n$ be a trace from s_0 to s_n and $\sigma = \alpha_0, \alpha_1, \ldots, \alpha_{n-1}$, where $\alpha_i \in \mathcal{I} \cup \mathcal{O} \cup \{\tau\}$, be a sequence of actions executed on the trace. We write $s_0 \xrightarrow{\sigma} s_n$, if there is such a trace, and call $Act(\sigma) \subseteq (\mathcal{I} \cup \mathcal{O})^*$ an *action word* that consists of the sequence of actions $\alpha_i \neq \tau$ executed in trace σ. We use ϵ to denote $Act(\tau^*)$. In the next definition transitions on action words are used to define extended STSs.

Definition 2 (Extended STS). *Given STS $\Sigma = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{I}, \mathcal{O}, \mathcal{R} \rangle$ its extended STS, written as $\hat{\Sigma} = \langle \mathcal{S}, \mathcal{S}_0, \mathcal{I}, \mathcal{O}, \hat{\mathcal{R}} \rangle$, $\hat{\mathcal{R}} \subseteq \mathcal{S} \times (\mathcal{I} \cup \mathcal{O})^* \times \mathcal{S}$ is defined as follows:*
for each pair of states s, s', s.t. $s \xrightarrow{\sigma} s'$, $(s, Act(\sigma), s') \in \hat{\mathcal{R}}$.

We say that a transition t of extended STS is *included by* some bigger transition t', written as $t \preceq t'$, if a trace described by the transition t' contains a trace described by t as a subsequence. For instance, the transition (s_1, ab, s_3) describing the trace s_1, a, s_2, b, s_3 is included by the transition (s_0, abc, s_4) describing the trace $s_0, \tau, s_1, a, s_2, b, s_3, c, s_4$.

The key idea underlying the introduced composition model is to provide an extended parallel product, where the synchronization is performed on *compatible* extended transitions.

Intuitively, two extended transitions $t_1 \in \hat{\mathcal{R}}^1$ and $t_2 \in \hat{\mathcal{R}}^2$ are *compatible*, written as $t_1 \approx t_2$, whenever they contain the same actions, even if we allow the order of actions in transitions to be different. The definition of compatibility relation depends on the particular communication model. It may require, for instance, that the matched symbols should appear in the same order, in the same places in words, etc. We will see the examples of this in the following section.

We define the product of extended state transition systems only for closed systems, that is all the communication actions of them should be shared. For the sake of simplicity we will introduce the definition of the product only for two STSs. The definition can be easily extended to the case with arbitrary numbers of components.

Definition 3 (Extended parallel product). *Let $\hat{\Sigma}^1$ and $\hat{\Sigma}^2$ be two extended STSs with $\mathcal{I}^1 = \mathcal{O}^2$ and $\mathcal{I}^2 = \mathcal{O}^1$. Their* extended product*, written $\hat{\Sigma}^1 \| \hat{\Sigma}^2$ is an extended STS defined as follows:*

- $\mathcal{S} = \mathcal{S}^1 \times \mathcal{S}^2$;
- $\mathcal{S}_0 = \mathcal{S}_0^1 \times \mathcal{S}_0^2$;
- $\mathcal{I} = \mathcal{O} = \emptyset$;
- $t = ((s_1, s_2), \epsilon, (s_1', s_2')) \in \hat{\mathcal{R}}$ *if*
  - $(s_1, \epsilon, s_1') \in \hat{\mathcal{R}}^1 \wedge s_2 = s_2'$, *or* $(s_2, \epsilon, s_2') \in \hat{\mathcal{R}}^2 \wedge s_1 = s_1'$;
  - $\exists\ t_1 = (s_1, \sigma_1, s_1') \in \hat{\mathcal{R}}^1$, $t_2 = (s_2, \sigma_2, s_2') \in \hat{\mathcal{R}}^2$, $t_1 \approx t_2$;

The transition relation in the definition includes two types of transitions. The first type describe actions where no communications appear (internal transitions). In transitions of second type each communication operation takes place at both sides (synchronized communication).

We remark that, due to the fact that the output actions are non-blocking in Web service interaction, the extended composition may represent an unfaithful model of the execution. Consider for instance the following modification of the cancellation mechanism (Fig. 5). VTA sends a cancellation to the Flight service and either receives a ticket from it, concluding that the cancellation is rejected, or a time-out occurs and it concludes that the cancellation can be performed. On the other side, the Flight service simply sends a ticket accepting then the cancellation. In this example there is a possibility for the VTA service to send a cancellation confirmation to the user even if the Flight service sends a ticket. It is easy to see that this scenario, which occurs in real executions, will not be present in the extended composition model. Therefore, verification results obtained on such model may be wrong as they do not consider all scenarios that can occur in real executions. In order to be able to figure out such situations we introduce the definition of *valid* extended parallel product. If the product is shown to be valid then it describes all possible scenarios and therefore is a faithful model of execution and can be safely used for further verification.

Intuitively, the situation where some messages can be emitted without being ever consumed should not occur in valid composition. We say that two extended transitions $t_1 \in \hat{\mathcal{R}}^1$ and $t_2 \in \hat{\mathcal{R}}^2$ are *partially compatible*, written as $t_1 \sim t_2$, if some output actions in one trace can be unmatched in the other trace.

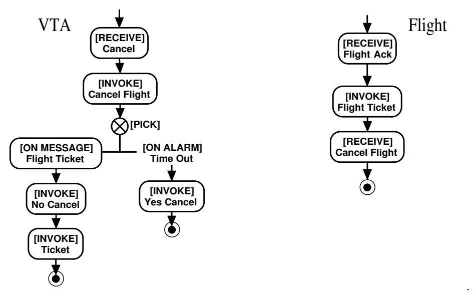


Fig. 5. Incorrect cancellation management

Definition 4 (Valid Composition). *Given two extended STSs $\hat{\Sigma}_1$ and $\hat{\Sigma}_2$, we say that their composition $\Sigma_1 \hat{\|} \Sigma_2$ is* valid *if for any state (s_1, s_2) reachable in the composition and any two transitions $t_1 = (s_1, \sigma_1, s_1')$ and $t_2 = (s_2, \sigma_2, s_2')$, such that $t_1 \sim t_2$, there are transitions $t_1' \succeq t_1$ and $t_2' \succeq t_2$, such that $t_1' \approx t_2'$.*

That is, the problem of checking the validity of the composition consists of finding reachable partially compatible extended transitions where some outputs can not be matched in any longer transitions.

5 Interpretation of Communication Models

We consider three instantiations of the general communication model introduced in the previous section. They correspond to the different cases of interactions introduced in Sect. 2. Formally, these models differ only in the way the actions in traces are matched when the compatibility is determined.

We remark that the extended composition model we introduce in this work relies on certain assumptions on the run-time environment. The following assumptions are common for any model we consider below:

– an output transition of the STS is non-blocking, i.e. the message can be emitted regardless a possibility to be ever consumed;
– the channels are perfect, i.e. the messages are not lost;
– the execution is fair, i.e. the enabled action can not be continuously ignored.

As we will show later, a particular model may additionally introduce certain specific assumptions.

5.1 Synchronous Communications

In this model the total order of all communication actions is relevant. That is, transitions are compatible if their sequences of communication actions are equivalent.

Definition 5 (Synchronous Compatibility). *Let $\hat{\Sigma}^1$ and $\hat{\Sigma}^2$ be two extended STSs, $t_1 = (s_1, \sigma_1, s'_1) \in \hat{\mathcal{R}}^1$ and $t_2 = (s_2, \sigma_2, s'_2) \in \hat{\mathcal{R}}^2$ extended transitions. Transitions t_1 and t_2 are compatible under synchronous communications model, written as $t_1 \approx_s t_2$ if $\sigma_1 = \sigma_2$.*

The validity of the system under this model coincides with the synchronizability property introduced in [8], i.e. the system is valid under this model whenever it does not allow for any concurrent message emissions. The example represented in Fig. 1 fits in this model. The composition of processes does not introduce any concurrent emissions, while the example in the Fig. 2 does. Therefore, the former can be faithfully verified under synchronous communication semantics, while the latter requires different semantic model, where the discovered kinds of executions might be considered as correct.

Due to the strong validity condition there is no need to put additional restrictions on the underlying middleware. Whenever the composition appears to be valid under this semantics, it can be executed independently of the BPEL engine implementation.

5.2 Ordered Asynchronous Communications

In the example of Fig. 2 there is a situation where the Flight and the VTA processes can send messages to each other simultaneously, thus violating the synchronous semantics. However, these messages are then consumed. Moreover, the mutual order of message emissions/consumptions is preserved in the composition. We position the systems of such kind as systems with ordered asynchronous communication semantics.

The compatibility relation for this class of systems has the following features. First, it handles each pair of partners separately. Second it distinguishes between the ordering of inputs from the ordering of outputs.

Definition 6 (Ordered Asynchronous Compatibility). *Let $\hat{\Sigma}^1$ and $\hat{\Sigma}^2$ be two extended STS, $t_1 = (s_1, \sigma_1, s'_1) \in \hat{\mathcal{R}}^1$ and $t_2 = (s_2, \sigma_2, s'_2) \in \hat{\mathcal{R}}^2$.*

Let $\omega_1^{\mathcal{I}}$ be the subsequence of σ_1, obtained by removing all actions symbols that are not input actions received from $\hat{\Sigma}^2$. Analogously, $\omega_1^{\mathcal{O}}$ is the subseqeunce of σ_1 with only outputs to $\hat{\Sigma}^2$.

Transitions t_1 and t_2 are compatible under ordered asynchronous communication model, written as $t_1 \approx_o t_2$, if $\omega_1^{\mathcal{I}} = \omega_2^{\mathcal{O}} \wedge \omega_2^{\mathcal{I}} = \omega_1^{\mathcal{O}}$.

This model is able to describe important scenarios, such as cancellation, that violate the synchronous communication semantics. It has to be noticed that the validity of the system under this model relies on the fact that the order in which messages are emitted has to be the same as the order in which they are consumed by the service. This, however, may be violated by real execution engines thus leading to incorrect behaviors. In order to avoid them, one should either verify that the system does not introduce incorrect behaviors under unordered communication model (see below), or enforce the order correctness explicitly in run-time. This can be done by introducing special monitors that will signal if the reordering has actually appeared.

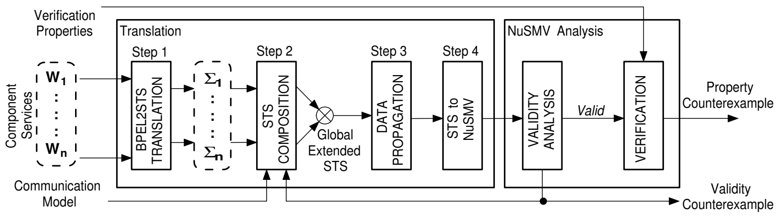


Fig. 6. The approach

5.3 Unordered Asynchronous Communications

While the previous model correctly describes the example in Fig. 2, it shows problems with the example in Fig. 3. The reason is that the latter will work correctly only if the order of messages is not relevant for consideration.

Such a model describes the systems where the order in which the messages are sent and received is irrelevant.

Definition 7 (Unordered Asynchronous Compatibility). *Let $\hat{\Sigma}^1$ and $\hat{\Sigma}^2$ be two extended STSs, $t_1 = (s_1, \sigma_1, s'_1) \in \hat{\mathcal{R}}^1$ and $t_2 = (s_2, \sigma_2, s'_2) \in \hat{\mathcal{R}}^2$.*

Transitions t_1 and t_2 are compatible under unordered asynchronous communication model, written as $t_1 \approx_u t_2$, if for any action symbol α appeared in σ_1 there is distinct corresponding action symbol appearing in σ_2.

This communication model is more liberal with respect to the previous in the sense that it permits more behaviors in the analyzed system. However, this requires a sophisticated queueing mechanism to be implemented in the BPEL engine.

6 Implementing the Approach

A preliminary prototype of a verification tool based on the parameric commuincation model presented in this paper has been implemented within the Astro toolkit and is available as part of the project (`http://www.astroproject.org`). Figure 6 represents the underlying architecture.

The tool consists of two modules. The first module, namely Translation module, is used to transform the initial set of BPEL process specifications into a specification accepted by the NuSMVmodel checker [6] for further analysis. There the input processes are transformed to the STS form; the extended product of their skeletons is built; the product is completed by adding the data manipulation operations and the result is emitted as NuSMVspecification. The specification is then passed to the analysis module, which verifies the specification. There the validity of the specification with respect to the given communication model is checked and the properties verification is then performed.

The algorithm that translates BPEL into NuSMVspecification relies on the following key consideration: in the extended composition it is not necessary to

Table 1. Verification results

Instance	Model	Translation	Validity	Deadlock	LTL
Example 1	Synchronous	0.5sec	1sec (valid)	0.5 sec	0.5sec
Example 2	Synchronous	2sec	4sec (invalid)	–	–
	Ordered	4sec	3sec (valid)	3sec	3sec
Example 3	Synchronous	4sec	5sec (invalid)	–	–
	Ordered	8sec	5sec (invalid)	–	–
	Unordered	9sec	5sec (valid)	5sec	4sec

consider all those extended transitions which contain as a prefix shorter transitions. I.e., if t_1 and t_2 are transitions of the extended composition, and $t_1 \preceq t_2$, then t_2 can be removed from the model without loosing behaviors. In the algorithm, we generate extended transitions incrementally, detect the compatibility as soon as it appears, and ignore longer transitions. This permits a finite system representation of the composed system and allows for efficient verification techniques to be applied. The only case when this approach may not work is when the composition contains cycles. In this case, we force a termination in the algorithm whenever an "incomplete" transition (in the sense of unmatched outputs) tries to traverse a cycle more than once. This condition may lead to consider invalid some scenarios that are actually valid. However, this problem does not appear in a wide range of interaction scenarios and models, including all non-cyclic protocols (such as those considered in Sect. 2), and all verifications based on the synchronous communication model. Currently we are working on a general solution for this problem.

We tested our approach on the different instantiations of VTA case study introduced in Sect. 2. The ranges of the domain types used in the messages (e.g. Flight, Time) were set to three values for each type. Although the examples described in the paper are relatively simple, they still are considerably more complex with respect to the set of samples presented in other tools (e.g. [8, 7]). The size of the (reachable) state space ranges in the examples from 500 to 2200 states. The results of the verification are summarized in the Table 1. For the valid systems also property verification was tested. Besides checking the models for deadlocks, we checked also a property, specified as a Linear-time Temporal Logic formula, that states that the User eventually finishes the process with success if and only if both the Flight and Hotel processes eventually succeed.

7 Related Work and Conclusions

In this paper we presented a unified framework for the analysis and verification of Web service compositions provided as BPEL specifications. The framework is based on the special form of composition, namely extended parallel product, that allows one to analyze whether the given system is valid under particular communication model. We have shown how different examples of composed systems require increasingly sophisticated communication models, which can be expressed in terms of out framework. The presented analysis approach allows one to iteratively check the validity of the given system against different communication models. Whenever the model is valid the actual verification of the system can be performed where different properties of interest can be checked.

The problem of analysis of communication systems with (potentially infinite) channels is widely studied in literature. Although the problem is undecidable in general [4], there are a lot of works on restricted subclasses of such systems for which certain problems were shown to be decidable (see e.g. [2]). In particular, an interesting class of systems that can be represented using Petri Nets formalism is widely used for analysis of asynchronous systems and workflows [14, 1]. Opposite to these works we investigate different cases of bounded models and make an attempt to prove the validity of the considered system under these models.

With respect to Web service analysis approaches, in particular BPEL processes, several works were described. The closest to our approach are the tools presented in [7] and [8]. The first one, namely LTSA-BPEL4WS, is based on the process algebra formalisms and allows for the analysis of basic properties of BPEL specifications, such as safety and progress checks. The tool currently does not support the analysis of composition of several BPEL specifications and was unable to handle complex specifications as those of the VTA case study. Moreover, it is based on the synchronous communications model thus being restrictive with respect to the set of systems it is able to correctly analyze. On the contrary, the WSAT tool [8] is equipped with the synchronizability analysis techniques that allow to check whether the behavior of the system is valid under synchronous communications semantics. However, the techniques currently provided allow only for partial analysis. That is, if the analyzed system does not pass the check it is not necessarily the case that the system is not synchronizable. The reason is that the synchronizability analysis is based on sufficient but not necessary conditions and that it ignores the information appearing in transitions conditions thus leading to spurious violations of the synchronizability. Also the provided techniques do not exceed the limits of the synchronizability analysis, and therefore do not allow for the reasoning about more sophisticated communication models.

One can also refer to works of [16], where the analysis is performed basing on Timed Automata, and of [5], inspired by process algebra notations. All these approaches exploit only the synchronous communication semantics, thus ruling out a certain class of systems (e.g. systems with cancellation), which are important in practice and can be managed in the proposed framework. On the contrary the aim of our approach is to attempt to find an appropriate communication model for the given system, under which it behaves correctly.

There are several directions for further research. We currently work on the extension of the translation from BPEL to STS for better coverage of constructs represented in the language. We also work on the optimizations of the validity analysis algorithm and enrichment of the approach with the possibility to reason about wider scope of communication models. Furthermore, we are interested in application of "knowledge level" reasoning techniques in order to perform the analysis of possibly infinite ranges of values and improve the verification performance shown in the previous section.

References

1. W. M. P. van der Aalst. Challenges in Business Process Management: Verification of Business Processing Using Petri Nets. *Bulletin of the EATCS 80: 174-199*, 2003.
2. P.A. Abdulla and B. Jonsson. Channel Representations in Protocol Verification (Preliminary Version). In *Proc. CONCUR'01*, August 2001.
3. T. Andrews, F. Curbera, H. Dolakia, J. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weeravarana. Business Process Execution Language for Web Services (version 1.1), 2003.
4. D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM, 2(5):323–342*, April 1983.
5. M. Koshkina and F. Breugel. Modelling and Verifying Web Service Orchestration by means of the Concurrency Workbench. *Proceedings of the Workshop on Testing, Analysis and Verification of Web Services (TAV-WEB), ACM SIGSOFT Software Engineering Notes*, 29(5), September 2004.
6. A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer (STTT)*, 2(4), 2000.
7. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of Web Service Compositions. In *Proc. ASE'03*, 2003.
8. X. Fu, T. Bultan, and J. Su. Analysis of Interacting BPEL Web Services. In *Proc. WWW'04*, 2004.
9. S. Graham, S. Simenov, T. Boubez, G. Daniels, D. Davis, Y. Nakamura, and R. Neyama. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL and UDDI.* Sams, 2001.
10. R. Khalaf, N. Mukhi, and S. Weerawarana. Service Oriented Composition in BPEL4WS. In *Proc. WWW'03*, 2003.
11. J. Koehler and B. Srivastava. Web Service Composition: Current Solutions and Open Problems. In *Proc. of ICAPS'03 Workshop on Planning for Web Services*, 2003.
12. S. Narayanan and S. McIlraith. Simulation, Verification and Automated Composition of Web Services. In *Proc. WWW'02*, 2002.
13. S. Nakajima. Model-checking verification for reliable web service. In *Proc. OOPSLA'02 Workshop on OOWS*, 2002.
14. J.L. Peterson. Petri Net Theory and the Modelling of Systems. *Prentice-Hall*, 1981.
15. M. Pistore, M. Roveri, P. Busetta. Requirements-Driven Verification of Web Services. In *Proc. WS-FM'04, ENTCS*, 2004.
16. P. Geguang, Z. Xiangpeng, W. Shuling, and Q. Zongyan. Towards the Semantics and Verification of BPEL4WS. In *Proc. WS-FM'04, ENTCS*, 2004.