



Communications semantics for WSBPEL Processes

Walid Fdhila and Mohsen Rouached and Claude Godart
LORIA-INRIA-UMR 7503
BP 239, F-54506 Vandœuvre-les-Nancy Cedex, France
{fdhilawa,rouached,godart}@loria.fr

Abstract

WSBPEL [2] opens up the possibility of applying a range of formal techniques to the verification of Web service behaviors from two points of view: constraints between activities within the same process and dependencies between activities of different processes. In a previous work, we have described an approach for the verification of Web service compositions defined by a set of BPEL processes. The key aspect of such a verification task is the model adopted for representing the communications among the services participating to the composition. In this paper, we propose to extend this approach to handle dependencies between activities of different process orchestrations through message exchanges. Our aim is to enable supporting models of service choreography with multiple interacting Web services compositions, from the perspective of a collaborative distributed composition development environment. The process of behavior analysis moves from a single local process to that of modelling and analyzing the behavior of multiple processes across composition domains.

1 Introduction

The ability to compose complex Web services from a multitude of available component services is one of the most important problems in service-oriented computing paradigm. Web service composition is the ability to aggregate multiple services into a single composite service that would provide a certain functionality, which otherwise cannot be provided by a single service.

While the technology for developing basic services and interconnecting them on a point-to-point basis has attained a certain level of maturity, there remain open challenges when it comes to engineering services that engage in complex interactions that go beyond simple sequences of requests and responses or involve large

numbers of participants.

In practice, there are two different (and competing) notions of modeling Web service compositions: orchestration and choreography. Orchestration describes how multiple services can interact by exchanging messages including the business logic and execution order of the interactions from the perspective of a single endpoint (i.e., the orchestrator). It refers to an executable process that may result in a persistent, multi step interaction model where the interactions are always controlled from the point of view of a single entity involved in the process. Choreography, on the other hand, provides a global view of message exchanges and interactions that occur between multiple process endpoints, rather than a single process that is executed by a party. Thus, choreography is more akin to a peer-to-peer (P2P) architecture and offers a means by which the rules of participation for collaboration are clearly defined and agreed upon. Even though there exists competing standards for both the models of composition, namely WS-BPEL [16] for orchestration and WS-CDL [3] for choreography, it is widely accepted that both orchestration and choreography can (and should) co-exist within one single environment.

Concerning WS-CDL, as discussed in [3], there are several places where its specification is not yet fully developed and a number of known issues remain open. Some issues of a more fundamental or practical nature are difficult to address and are likely to require a significant review of the language's underlying meta-model and implied techniques. These issues primarily stem from three factors: (i) lack of separation between meta-model and syntax, (ii) lack of direct support for certain categories of use cases and, (iii) lack of comprehensive formal grounding (see [3] for details).

On the contrary, BPEL is quickly emerging as the language of choice for Web service composition. It opens up the possibility of applying a range of formal techniques to the verification of the behavior of Web services (see, e.g. [11, 13, 20]). For instance, it

is possible to check the internal business process of a participant against the external business protocol that the participant is committed to provide; or, it is possible to verify whether the composition of two or more processes satisfies general properties (such as deadlock freedom) or application-specific constraints (e.g., temporal sequences, limitations on resources). These kinds of verifications are particularly relevant in the distributed and highly dynamic world of Web services, where each partner can autonomously redefine business processes and interaction protocols.

However, one common problem of the different techniques adopted is related to the model used 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, the semantics for how to translate the connectivity and communication between activities of the partner processes rather than from a single process focus are not taken into account.

To address these shortcomings, we propose in this paper a semantic framework that provides a foundation for addressing the above limitations by supporting models of service choreography with multiple interacting Web services compositions, from the perspective of a collaborative distributed composition development environment. The process of behaviour analysis moves from a single local process to that of modelling and analyzing the behavior of multiple processes across composition domains. We show also how to translate the connectivity and communication between activities of the partner processes rather than from a single process focus. These may also contain communication actions or dependencies between communication actions that do not appear in any of the service's behavioral interface(s). This is because behavioral interfaces may be made available to external parties, and, thus, they should only show the information that actually needs to be visible to these parties.

The remainder of this paper is structured as follows. Section 2 describes the background and the issues involved in Web service compositions verifications. In Section 3, we detail our approach and explain the different steps for getting our communication model. The implementation of the approach is discussed in Section 4. Finally, Section 5 summarizes the ideas explained in the paper and outlines some future directions.

2 Background

Standards for service composition cover three different, although overlapping, viewpoints: Choreography, Behavioral interface (also called abstract process in BPEL), and Orchestration (also called executable process in BPEL).

While a choreography model describes a collaboration between a collection of services in order to achieve a common goal, an orchestration model describes both the communication actions and the internal actions in which a service engages. Internal actions include data transformations and invocations to internal software modules (e.g., legacy applications that are not exposed as services). An orchestration may also contain communication actions or dependencies between communication actions that do not appear in any of the service's behavioral interface(s). This is because behavioral interfaces may be made available to external parties, and, thus, they should only show the information that actually needs to be visible to these parties.

With respect to Web service analysis approaches, in particular BPEL processes, several works were described to capture the behavior of BPEL [1] in some formal way. Some advocate the use of finite state machines [10], others process algebras [9], and yet others abstract state machines [8] or Petri nets [19, 18, 23]. But they mainly focus on introducing a semantic discovery service and facilitating semantic translations. Other attempts to formalize BPEL specification and a detailed comparison between them can be found in [25, 24]. [24] is a tutorial that provides an overview of the different models of BPEL that have been proposed. Furthermore, the authors discuss the verification techniques for BPEL that have been put forward and the verification tools for BPEL that have been developed.

In terms of choreography and Web service conversations, work on asynchronous Web service communication has been described in [13, 12], with an example focus on the BPEL4WS specification reported in [13]. A formal specification framework is described to analyze the conversations proposed by the asynchronous communication channels utilized on the Internet. The technique proposed appears more useful for modelling general Web service communications, rather than that of compositional specifics. Both the work on asynchronous and BPEL4WS interaction modelling is achieved through the use of Guarded Finite State Automata (GFSAs) which enables data dependencies to be modeled alongside process transitions. In [6] the authors describe an approach to formalizing conversations, by way of mapping the WSCI standard to CCS for Web service choreography descriptions. The tech-

nique is similar to that of formalizing compositions by way of mapping each of the actions and data parameters between two or more partnered services in choreography. The conversation is traced by modelling the Web service invocations with that of the receive and reply actions of the partnered service. The authors call for a common view of representing both composition and choreography models, such that fluid design and maintenance of individual specifications are not detrimental to the development effort.

[14] describes 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.

Berardi and al. [5, 4] also provide a formal framework where services are represented using transition systems. The approach assumes that the services exchange messages according to a pre-defined communication topology (referred to as the linkage structure), which is expressed as a set of channels.

Manolescu and al. [17] present a high-level language and methodology for designing and deploying Web applications using Web services. In particular, the authors extend WebML [7] to support message-exchange patterns present in WSDL and use the WebML hypertext model for describing Web interactions and defining specific concepts in the model to represent Web service calls. Consequently, the Web service invocation is captured by a visual language representing the relationships between the invocations and the input/output messages.

A common pattern of the above attempts is that the orchestration and the choreography are not usually expressed within one single environment and therefore the verification techniques must be modified before using them. Instead, in our research work, we aim to provide a uniform framework that is capable of addressing this shortcoming by providing a guide on how to translate the semantics of the BPEL specification to EC and map implementation abstractions which preserve the interaction behaviour between services, yet also disposing of process characteristics which are not required in the analysis. Then, we elaborated these models to analyze the conversations of compositions across choreography scenarios, providing both interface and behavioral compatibility verification processes.

3 Communication semantics for WS-BPEL processes

To illustrate our ideas, we refer to a running example implemented as a BPEL process realizing a Car Rental Agency service (a complete description can be found in [21]). It interacts with a Car Broker Service (CRS), which controls the operations of the branch; with a User Interaction Service (UIS), through which customers can make car rental requests; with a Car Information Service (CIS), which maintains a database of cars availability and allocate cars to customers; with a Car Park Sensor Service, which exposes as a Web service the sensor that senses cars as they are driven in or out of the car park of the branch. Each of the component services can also be implemented as a BPEL process since it needs some other processes to ensure its role in the collaboration.

3.1 Event-driven specification

Given the fact that we consider communications actions where ordering and timing are relevant and we adopt an event driven reasoning, the Event Calculus (EC) [15] seems to be a solid basis to start from. Another key element of this choice is that orchestration and choreography should co-exist within one single environment, and in our case the orchestration verification framework is based on EC logic.

EC is a temporal formalism based on a first order logic, that can be used to specify the *events* that appear within a system and the effect (or the *fluents*) of these events. It includes an explicit *time structure* that dates the system changes caused by the occurrence of the events. EC includes the predicates *Happens*, *Initiates*, *Terminates* and *HoldsAt*, as well as some auxiliary predicates defined in terms of these. *Happens(a, t)* indicates that event (or action) *a* actually occurs at time-point *t*. *Initiates(a, f, t)* (resp. *Terminates(a, f, t)*) means that if event *a* were to occur at *t* it would cause fluent *f* to be *true* (resp. *false*) immediately afterwards. *HoldsAt(f, t)* indicates that fluent *f* is true at *t*. The auxiliary predicate *Clipped(t1, f, t2)* expresses whether a fluent *f* was terminated during a time interval $[t1, t2]$.

To formally specify and reason about the interactions between a set of BPEL processes, we use four different types of events showed in Figure 1.

1. *invoke_input*: The invocation of an operation by the composition process of the system in one of its partner services. The term *invoke.ic(PartnerService, Op(oId, inVar))* represents the invocation event. In this term,

Type	Event
invoke_input	Happens (invoke_ic(PartnerService,Op(oId,inVar)),t)
invoke_output	Happens (invoke_ir(PartnerService,Op(oId)),t)
receive	Happens (invoke_rc(PartnerService,Op(oId)),t)
reply	Happens (reply(PartnerService,Op(oId,outVar)),t)

Figure 1. Events expressed in Event Calculus

Op is the name of the invoked operation, $PartnerService$ is the name of the service that provides Op , oId is a variable whose value determines the exact instance of the invocation of Op within a specific instance of the execution of the composition process, and $inVar$ is a variable whose value is the value of the input parameter of Op at the time of its invocation.

2. *invoke_output*: The return from the execution of an operation invoked by the composition process in a partner service. The term $invoke_ir(PartnerService,Op(oId))$ in this predicate represents the return event. $PartnerService$, Op and oId in this term are as defined in (1). In cases where Op has an output variable $outVar$, the value of this variable at the return of the operation is represented by the predicate: **Initiates**($invoke_ir(PartnerService,Op(oId)), equalTo(outVar1, outVar), t$). This predicate expresses the initialization of a fluent variable ($outVar1$) with the value of $outVar$. The fluent $equalTo(VarName, val)$ signifies that value of $VarName$ is equal to val .
3. *receive*: The invocation of an operation in the composition process by a partner service. The term $invoke_rc(PartnerService,Op(oId))$ in this predicate represents the invocation event. Op and oId are as defined in (1) and $PartnerService$ is the name of the service that invokes the operation. In cases where Op has an input variable $inVar$, the value of this variable at the time of its invocation is represented by the predicate **Initiates**($invoke_rc(PartnerService,Op(oId)), equalTo(inVar1, inVar), t$). This predicate expresses the initialization of a fluent variable $inVar1$ with the value of $inVar$.
4. *reply*: The reply following the execution of an operation that was invoked by a partner service in the composition process. The term $reply(PartnerService,Op(oId,outVar))$ in this

predicate represents the reply event. In this term, Op and oId are as defined in (1), $PartnerService$ is the name of the service that invoked Op , and $outVar$ is a variable whose value is the value of the output parameter of the operation at the time of the reply.

3.2 The approach

As mentioned so far, our objectif is to provide a model of service choreography with multiple interacting Web services compositions, from the perspective of a collaborative distributed composition development environment. The process of behaviour analysis moves from a single local process to that of modelling and analyzing the behaviour of multiple processes across composition domains. We look also for translating the connectivity and communication between activities of the partner processes rather than from a single process focus (see Figure 2). These may also contain communication actions or dependencies between communication actions that do not appear in any of the service's behavioral interface(s). In this section, we discuss how to realize this objective.

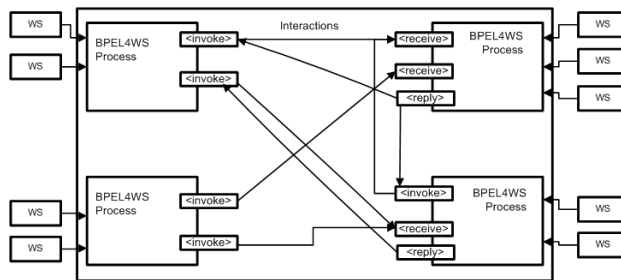


Figure 2. Web Service Compositions Interactions

To start, we require a process to analyze which activities are partnered in the composition. For example, *invoke* from the UIS service (a rental request) will be received by the CRS process (*receive* a rental request). Equally, the CRS *invoke* activity, to check the availability of cars by contacting CIS, will be aligned with *receive* in the CRS process. In WSBPEL, the communication is based upon a protocol of behavior for a local service. However, the partner communication can concisely be modeled using the synchronous event passing model, described in [16]. The Sender-Receiver example discussed uses *Channels* to facilitate message/event passing between such a sender and receiver model. The representation of a channel in WSBPEL is known as a

port. The significant element of this discussion used in our process is that of synchronization of the invoking and receiving events within compositions between ports and whether this has been constructed concurrently (*flow* construct in WSBPEL) or as a sequence (*sequence* construct in WSBPEL) of activities.

In the following, we seek to further our modelling of WSBPEL interactions through two viewpoints. First, we examine the interactions within the choreography layer of Web service compositions collaborating in a global goal. Secondly, through further behaviour analysis, we model the interaction sequences built to support multiple-partner conversations across enterprise domains and with a view of wider goals.

Our approach relies on four steps: (1) identifying services conversations, (2) identifying partners involved in the composition and their respective roles, (3) linking composition interactions by revealing the invocation style, points at which interaction occurs and linking between partners using port connectors. We introduce, also, the interaction modelling algorithm, we proposed, in details and (4) building interaction models using our formalism.

3.3 Service conversations

Events exchange is a basic concept of Web service composition interactions. In this sense, Web service modelling involves interactions and their interdependencies description from structural and behavioral point of view. In this step, we mainly identify conversations between two or more participants. Note that a service may be engaged simultaneously in several conversations with different partners. A conversation defines how interactions can start and end depending on the goal of conversation. It specifies also the order in which several scenarios could occur.

To model these conversations in the context of several Web service compositions, we perform an analysis process on all the implementation processes and use an algorithm as part of this analysis to semantically check and link partner process interactions. The algorithm takes as inputs the partner service interfaces (WSDL documents) and the implementation models (BPEL documents). The output of this phase is a list of interaction activities.

3.4 Service partners and roles

An important requirement for realistic modelling of business processes is the ability to model the required relationship with a partner process. WSDL already describes the functionality of a service provided by a

partner, at both the abstract and concrete levels. The relationship of a business process to a partner is typically peer-to-peer, requiring a two-way dependency at the service level. In other words, a partner represents both a consumer of a service provided by the business process and a provider of a service to the business process. In this sense, a partner may be considered to have one or many roles depending on what behaviour the partner's service provides. The role indicator is used primarily to distinguish what the business process is referencing as part of the collaborative business process.

3.5 Linking composition interactions

To model interacting Web service compositions there is clearly a need to elaborate our analysis of implementations by linking compositional interactions based upon: (i) activities within the process (identifying invocation style (*rendez vous* or *request only*), identifying and recording the points at which interaction occurs), (ii) the abstract interface (linking between the private process activities and the public communication interface declared in the abstract WSDL service description).

To model the semantics of linking interactions between processes requires a mapping between activities in each of the processes translated and building an event port connector for each of the interaction activities linking *invoke* (input) with *receives*, and *replies* (output) with the returned message to an *invoke*.

Before introducing our choreography modelling algorithm, we define some formal notations useful for its understanding.

3.5.1 Algorithm

Definitions

- 1- Let O be the set of all operations provided by a Web service in the choreography.
- 2- Let C_w be the BPEL process of the partner W .
- 3- A BPEL process C_{wi} is a quadruple (In, P, A, W_i) where
 - $In \subset O$ represents the WSDL process interface: $In = \{w_i.o_n \mid 0 \leq n \leq n_{wi}\}$
 - W_i the set of partners defined in the process C_{wi}
 - $P \subset O$ the set of the operations of partner w_j of w_i ($j \in I$), such as $P = \{w_j.o \mid w_j \neq w_i \text{ and } \exists j \in I, w_j.o \in In_j\}$

- A is the set of the invocation activities such as $\forall a \in A$:
 - * $a.o$ represents the invoked operation
 - * $a.p$ represents the invoked partner

Algorithm The physical linking of partnerlinks, partners and process models is undertaken as follows. For each invocation in a process, a messaging port is created. WSBPEL defines communication in a synchronous messaging model. WSBPEL process instance support in the specification specifies that in order to keep consistency between process activities, a synchronous request mechanism must be governed. The synchronous model can be formed by the following process.

Algorithm 1: Interactions modelling algorithm

```

for each Composition  $C_{w_i}$  do
  for each  $a \in A_{w_i}$  do
     $P\_Local \leftarrow a.p$ 
     $P\_link \leftarrow P\_local.partnerLink$ 
     $PLT \leftarrow P\_link.partnerLinkType$ 
     $Port\_Type \leftarrow PLT.portType$ 
    for each  $In_{w_j}$  ( $w_j \in W_i$ ) do
      if  $In_{w_j}.porttype = Port\_Type$  then
         $actual\_partner \leftarrow w_j$ 
        Lookup  $w_j.o \in P_{w_j}$  such as
         $w_j.o = a.o$ 
      if  $a.o.output$  is in (rendez-vous style)
      then
        Add invokeOutput action to activity model
        Build reply-invokeOutput connector
      Build invoke-receive connector

```

For every composition process selected for modelling we extract all the interaction activities in this process. Interaction activities are service operation invocations (requests), receiving operation requests and replying to operation requests. In addition to an invocation request, we also add an invocation reply to synchronize the reply from a partner process with that of the requesting client process. The list is then analyzed for invocation requests, and for each one found a partner/port lookup is undertaken to gather the actual partner that is specified in a partnerlink declaration. To achieve this, a partner list is used and the partner referenced in the invocation request is linked back to a partnerlink reference. The partnerlink specifies the porttype to link operation and partner with an actual

interface definition. To complete the partner match, all interface definitions used in composition analysis are searched and matched on porttype and operation of requesting client process. This concludes the partner match. A port connector bridge is then built to support either a simple request invocation (with no reply expected) or in “rendez-vous” style, building both invoke/receive and reply-invoke_output models. This supports the model mapping. The sequence is then repeated for all other invocations in the selected composition process, and then looped again for any other composition processes to analyze. We therefore specify an algorithm that will enable mechanical linking between activities, partners and process compositions. The algorithm supports a mechanical implementation of linking composition processes together based upon their interaction behaviour. Two build phases are required as part of the algorithm, being that of building a reply-invoke_output port and invoke-receive connector between partnered processes.

In summary, the algorithm described provides a port connector based implementation of the communication between two partner processes. Where multiple partners communication is undertaken in a composition, a port connector is built between each instance of a message (and optionally a reply if used in rendez-vous interaction style). In the following, we explain how to construct our port connector model.

3.6 Building interaction models

The activity of building port connectors for our integration mapping is based on the basic concept of event passing in the formation of Web service composition communication. The essence of this work is that events are passed through channels. A channel connects two and only two processes, in which a single process can receive from a channel. The term channel is used to symbolize that an one-to-one channel is used in process synchronization. A connector is the implementation between port and channel, in that a sender port is connected to a sender-receiver channel.

3.6.1 Event Invocations Connectors

To build connected composition interactions, port connector channels are used for each of the invocation styles between two or more partnered compositions. The algorithm is used from the viewpoint of a process composition at the “center of focus”, that is, the one in which initial process analysis is being considered. The interface of subsequent partner interactions is used in the algorithm to obtain a link between two

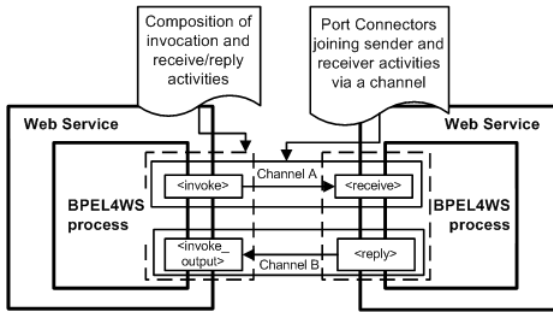


Figure 3. Channels and Interaction Activities of Web Service Compositions

partners and an actual operation. For example in Figure 3, two WSBPEL processes interact using both a request only invocation (Channel A) and a Rendez-vous style (Channel A and B). Our model of interactions using channels takes into consideration both synchronous and asynchronous interactions between partners. The model produced from analysis of the compositions is from the viewpoint of the composition performing as part of a role in choreography. This makes the model providing an abstract view of interactions for the purpose of linking invocations and not on the actual order of messages received by the process host architecture (synchronous and asynchronous messaging models for Web services can be referred in [13]).

Request only invocation (Channel A) Web service compositions specified with the *invoke* construct and only an *input* container attribute declare an interaction on a request only basis (there is no immediate reply expected). More generally this requirement is for a reliable message invocation without any output response from the service host (other than status of receiving the request). The model for this is illustrated in Figure 4.

Rendezvous style invocation (Channels A and B) “Rendezvous” (Request and Reply) invocations are specified in WSBPEL with the *invoke* construct, with both *input* and *output* container attributes. To model these types of interactions, we use a generic port model for each process port. A synchronous event model in Web services compositions (such as WSBPEL) requires an additional activity of an “input_output” to link a reply in a partnered process to that of the caller receiving the output of the invoke, however, this is necessary only if the invocation style is that of rendez-vous. The event synchronization for

WS Interaction	Port Action	BPEL Actions (Example)
Invoke (client)	Invoke_input	invoke_client_CRS_CarRequest
Receive (Partner)	Receive	Receive_client_CRS_CarRequest
Reply (Partner to client)	Reply	reply_CRS_client_CarRequest
	Invoke_output	output_CRS_client_CarRequest

Table 1. Mapping Process Activities to Port Connectors

this port model is shown in Figure 5.

Coming back to the CRS example introduced so far, Figure 6 shows an interaction scenario to illustrate how the previous interactions can be established.

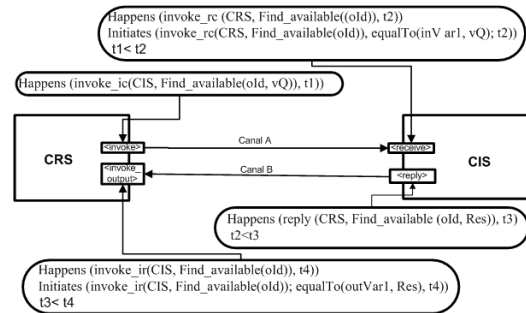


Figure 6. Event Invocation Connectors

3.6.2 Mapping Process Activities to Port Connectors

The next step in the port connector modelling process is to map the activities of the WSBPEL process to the port connector activities. This is achieved using the semantics of WSBPEL for the interaction activities discussed earlier and replacing the port connector activities appropriately.

The *invoke* activity in BPEL4WS is mapped from the client process to the *invoke_input* action of the port connector - this represents the initial step of a request between Web service partners.

The associated receiving action of the WSBPEL partner process is mapped to the *receive* activity in the port connector. The reply from the partner process to the client process is mapped to the *reply* in the partnered process. Both *receive* and *reply* activities in the WSBPEL are discovered as part of the interface analysis described before. Table 1 lists the mapping explained here.

$$\begin{aligned}
& \forall(t1:time)\mathbf{Happens}(invoke_ic(PartnerService, Operation(oId, inVar)), t1) \implies \\
& (\exists t2)\mathbf{Happens}(invoke_rc(PartnerService, Operation(oId)), t2) \wedge \\
& \mathbf{Initiates}(invoke_rc(PartnerService, Operation(oId)), equalTo(inVar1, inVar), t2) \wedge (t1 < t2). \\
& \forall(t2:time)\mathbf{Happens}(invoke_rc(PartnerService, Operation(oId)), t2) \wedge \\
& \mathbf{Initiates}(invoke_rc(PartnerService, Operation(oId)), equalTo(inVar1, inVar), t2) \implies \\
& (\exists t1)\mathbf{Happens}(invoke_ic(PartnerService, Operation(oId, inVar)), t1) \wedge (t1 < t2).
\end{aligned}$$

Figure 4. Request only invocation

$$\begin{aligned}
& \forall(t1:time)\mathbf{Happens}(invoke_ic(PartnerService, Operation(oId1, inVar)), t1) \implies \\
& (\exists t2)\mathbf{Happens}(invoke_rc(PartnerService, Operation(oId1)), t2) \wedge \\
& \mathbf{Initiates}(invoke_rc(PartnerService, Operation(oId1)), equalTo(inVar1, inVar), t2) \wedge (t1 < t2). \\
& \forall(t2:time)\mathbf{Happens}(invoke_rc(PartnerService, Operation(oId)), t2) \wedge \\
& \mathbf{Initiates}(invoke_rc(PartnerService, Operation(oId)), equalTo(inVar1, inVar), t2) \implies \\
& (\exists t1)\mathbf{Happens}(invoke_ic(PartnerService, Operation(oId, inVar)), t1) \wedge (t1 < t2). \\
& \forall(t3:time)\mathbf{Happens}(reply(PartnerService, Operation(oId2, outVar)), t3) \implies \\
& (\exists t4)\mathbf{Happens}(invoke_ir(PartnerService, Operation(oId2)), t4) \wedge \\
& \mathbf{Initiates}(invoke_ir(PartnerService, Operation(oId2)), equalTo(outVar1, outVar), t4) \wedge (t3 < t4). \\
& \forall(t4:time)\mathbf{Happens}(invoke_ir(PartnerService, Operation(oId2)), t4) \wedge \\
& \mathbf{Initiates}(invoke_ir(PartnerService, Operation(oId2)), equalTo(outVar1, outVar), t4) \implies \\
& (\exists t3)\mathbf{Happens}(reply(PartnerService, Operation(oId2, outVar)), t3) \wedge (t3 < t4).
\end{aligned}$$

Figure 5. Rendez-vous style invocation

4 Implementation

As a verification back-end, we have used an automated induction-based theorem prover SPIKE [22]. More details about the verification process and the encoding ingredients can be found in [22].

Then, to support the choreography aspects introduced in this paper, we have extended our BPEL2EC tool presented in [21]. The BPEL2EC tool is built as a parser that can automatically transform a given WS-BPEL process into EC formulas according to the transformation scheme. It takes as input the specification of the Web service composition as a set of coordinated Web services in WSBPEL and produces as output the behavioral specification of this composition in Event Calculus. The description of this implementation is beyond the scope of this paper and may be found in [21].

However, this tool already supports the orchestration aspect model of BPEL and does not provide any support for the choreography level. In this section, we would like to focus on the basic structure of our extension to the BPEL2EC tool depicted in Figure 7.

The tool was developed using Java programming language. Since BPEL and WSDL are xml specifications, we have implemented two xml parsers. The both were developed using the application programming interface JDOM (Java Document Object Model).

The starting point is a set of Web service com-

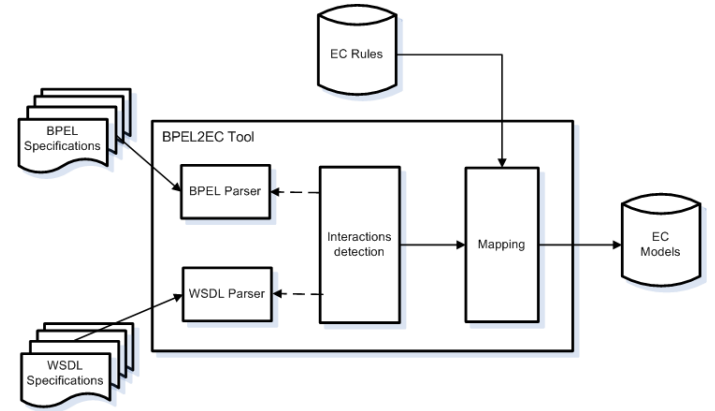


Figure 7. Basic structure of BPEL2EC tool

positions specifications in BPEL and all interfaces of the Web services participating in the collaboration. Interactions detection module serves to reveal all inter-compositions interactions using BPEL and WSDL parsers. The output of this step is a set of all peer-to-peer relationships between the actual partners. The mapping step, use the EC translation rules defined in section 3.6 to model interactions previously identified and build port connectors between every two interacting partners. Those models are saved into log files which will be useful for both verification and validation by measuring the actual run time deviation with

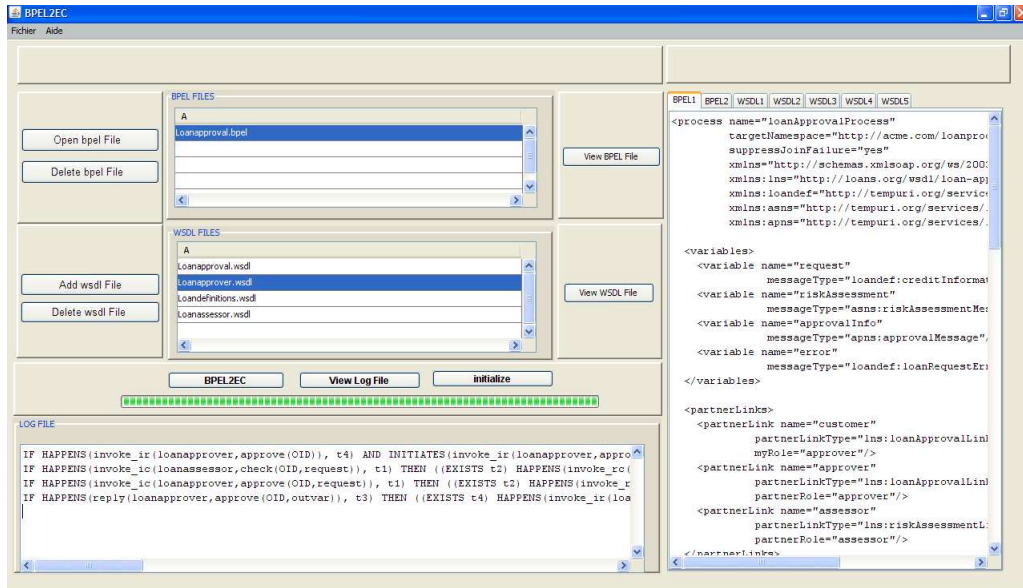


Figure 8. A screenshot of the BPEL2EC tool

respect to the models.

Figure 8 shows a snapshot of BPEL2EC in action. The BPEL and WSDL specifications are loaded into BPEL2EC tool which generates the formal models in terms of rules expressed in EC language. Note that the tool saves automatically the results in log files. This enables the designer to check the translation process.

5 Conclusion

In this paper, we have described an elaboration of composition models to support a view of interacting Web service composition processes extending the mapping from BPEL4WS to EC discussed in our previous work [22], and introducing Web service interfaces for use in modelling between services. The ability to model these conversations is important to discovering how Web service interactions fulfill a choreography scenario and if the conversation protocol implement is compatible with that of partnered services. In essence, our view of modelling has moved from analyzing a local process, or in other word a single composition, with that of other services and their interactions. We have also extended the BPEL2EC tool to support multiple process conversations as an implementation of our approach.

References

- [1] T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.
- [2] A. Arkin, S. Askary, B. Bloch, and F. Curbera. Web services business process execution language version 2.0. Technical report, OASIS, December 2004.
- [3] A. Barros, M. Dumas, and P. Oaks. Critical overview of the web services choreography description language (ws-cdl), March 2005. <http://www.bptrends.com>.
- [4] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, M. Lenzerini, and M. Mecella. Modeling data processes for service specifications in colombo. In M. Missikoff and A. D. Nicola, editors, *EMOI-INTEROP*, volume 160 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.
- [5] D. Berardi, D. Calvanese, G. D. Giacomo, R. Hull, and M. Mecella. Automatic composition of web services in colombo. In A. Cal, D. Calvanese, E. Franconi, M. Lenzerini, and L. Tanca, editors, *SEBD*, pages 8–15, 2005.

- [6] A. Brogi, C. Canal, E. Pimentel, and A. Vallecillo. Formalizing web service choreographies. *Electr. Notes Theor. Comput. Sci.*, 105:73–94, 2004.
- [7] S. Ceri, P. Fraternali, and A. Bongio. Web modeling language (webml): a modeling language for designing web sites. *Comput. Netw.*, 33(1-6):137–157, 2000.
- [8] D. Fahland and W. Reisig. ASM-based semantics for BPEL: The negative control flow. In D. Beauquier and E. Börger and A. Slissenko, editor, *Proc. 12th International Workshop on Abstract State Machines*, pages 131–151, Paris, France, March 2005.
- [9] A. Ferrara. Web services: A process algebra approach. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 242–251, New York, NY, USA, 2004. ACM Press.
- [10] J. Fisteus, L. Fernández, and C. Kloos. Formal verification of BPEL4WS business collaborations. In K. Bauknecht, M. Bichler, and B. Proll, editors, *Proceedings of the 5th International Conference on Electronic Commerce and Web Technologies (EC-Web '04)*, volume 3182 of *Lecture Notes in Computer Science*, pages 79–94, Zaragoza, Spain, Aug. 2004. Springer-Verlag, Berlin.
- [11] H. Foster, J. Kramer, J. Magee, and S. Uchitel. Model-based verification of web service compositions. In *18th IEEE International Conference on Automated Software Engineering (ASE)*, 2003.
- [12] X. Fu. Formal Specification and Verification of Asynchronously Communicating Web Services. Phd Thesis, Santa Barbara, CA, USA, University of California, 2004.
- [13] X. Fu, T. Bultan, and J. Su. Analysis of interacting bpel web services. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 621–630, New York, NY, USA, 2004. ACM Press.
- [14] R. Kazhamiakin, M. Pistore, and L. Santuari. Analysis of communication models in web service compositions. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 267–276, New York, NY, USA, 2006. ACM.
- [15] R. Kowalski and M. J. Sergot. A logic-based calculus of events. *New generation Computing 4(1)*, pages 67–95, 1986.
- [16] J. Magee and J. Kramer. *Concurrency: state models & Java programs*. John Wiley & Sons, Inc., New York, NY, USA, 1999.
- [17] I. Manolescu, M. Brambilla, S. Ceri, S. Comai, and P. Fraternali. Model-driven design and deployment of service-enabled web applications. *ACM Trans. Inter. Tech.*, 5(3):439–479, 2005.
- [18] A. Martens. Analyzing Web Service Based Business Processes. In M. Cerioli, editor, *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering (FASE 2005)*, volume 3442 of *Lecture Notes in Computer Science*, pages 19–33. Springer-Verlag, Berlin, 2005.
- [19] C. Ouyang, W. Aalst, S. Breutel, M. Dumas, , and H. Verbeek. Formal Semantics and Analysis of Control Flow in WS-BPEL. BPM Center Report BPM-05-15, BPMcenter.org, 2005.
- [20] M. Pistore, M. Roveri, and P. Busetta. Requirements-driven verification of web services. *Electr. Notes Theor. Comput. Sci.*, 105:95–108, 2004.
- [21] M. Rouached, W. Gaaloul, W. M. P. van der Aalst, S. Bhiri, and C. Godart. Web service mining and verification of properties: An approach based on event calculus. In *Proceedings 14th International Conference on Cooperative Information Systems (CoopIS 2006)*, November 2006.
- [22] M. Rouached and C. Godart. Requirements-driven verification of wsbpel processes. In *Proceedings of the IEEE International Conference on Web Services (ICWS'07)*. Salt Lake City, Utah, USA, July 9-13 2007.
- [23] C. Stahl. Transformation von BPEL4WS in Petrinetze (In German). Master's thesis, Humboldt University, Berlin, Germany, 2004.
- [24] F. van Breugel and M. Koshkina. Models and verification of bpel. Available at <http://www.cse.yorku.ca/franck/research/drafts/tutorial.pdf>, 2006.
- [25] Y. Yang, Q. Tan, and Y. Xiao. Verifying web services composition based on hierarchical colored petri nets. In *IHIS '05: Proceedings of the first international workshop on Interoperability of heterogeneous information systems*, pages 47–54, New York, NY, USA, 2005. ACM Press.