Web Services Compositions Modelling and Choreographies Analysis

Mohsen Rouached, Walid Fdhila, Claude Godart

Web Services Compositions Modelling and Choreographies Analysis
Extension from ICWS 2008 paper id: 505

Mohsen Rouached*, Walid Fdhila**, and Claude Godart**

* Laboratoire d'InfoRmatique en Image et Systèmes d'information (LIRIS)
CNRS UMR 5205, France
mohsen.rouached@liris.cnrs.fr

**LORIA-INRIA
BP 239, F-54506 Vandoeuvre-lès-Nancy Cedex, France
{Fdhila, godart}@loria.fr

ABSTRACT:

In (Rouached, Godart and al. 2006; Rouached, Godart 2007), we have described the semantics of WSBPEL by way of mapping each of the WSBPEL (Arkin, Askary and al. 2004) constructs to the EC algebra and building a model of the process behaviour. With these mapping rules, we have described a modelling approach of a process defined for a single Web service composition. However, this modelling is limited to a local view and can only be used to model the behaviour of a single process. A series of compositions in Web service choreography need specific modelling activities that are not explicitly derived from an implementation. An elaboration of modelling is then required to represent the behaviour of interacting compositions across partnered processes. This elaboration provides a representation that enables us to perform analysis of service interaction for behaviour properties. The ability to perform verification and validation between execution and design, and within the process compositions themselves, is a key requirement of the Web services architecture specification. In this paper, we further the semantic mapping to include Web service composition interactions through modelling Web service conversations and their choreography. We describe this elaboration of models to support a view of interacting Web service compositions extending the mapping from WSBPEL to EC, and including Web service interfaces (WSDL) for use in modelling between services. The verification and validation techniques are also exposed. An automated induction-based theorem prover is used as verification back-end.

KEY WORDS:
Web service composition, Orchestration, choreography, Verification and Validation

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, WSBPEL for orchestration and WS-CDL (Barros, Dumas and al. 2005) 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 (Barros, Dumas and al. 2005), 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 (Barros, Dumas and al. 2005) for details).

On the contrary, WSBPEL 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 Web services behaviour (see, e.g. (Foster, Uchitel and al. 2003; Fu, Bultan and al. 2004; Pistore, Roveri and al. 2004)). 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 WSBPEL execution engines is both very complex and implementation dependent. More precisely, WSBPEL 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 WSBPEL 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. In Section 2, we discuss the Web services composition modelling aspects. Three viewpoints and the relationships between them are presented. Section 3 details our approach of modelling services choreographies and explains the different steps for getting our communication model. A running example is used to illustrate the main ideas. The verfication and validation aspects are discussed in Section 4. More precisely, this section introduces the verfication techniques that we have used, the ingredients of our encoding, and the implementation of the model. Section 5 describes the related work, and outlines where our work is positioned alongside these. Finally, Section 6 summarizes the ideas explained in the paper and outlines some future directions.

WEB SERVICE COMPOSITIONS MODELLING

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

Choreography is typically initiated by an external source (a client or service) and ends with a target service or a reply to the source. Such interactions during this choreography pose questions such as; *can messages be sent and received in any order? What are the rules governing the sequencing of messages?* And can a global view of the overall exchange of messages be drawn? *Can we verify, modify and monitor the behaviour?*

The viewpoints presented above have some overlap (Dijkman, Dumas 2004). This overlap can be exploited within service composition methodologies to perform consistency checks between viewpoints or to generate code. For example, a choreography model can be used to generate the behavioral interface that each participating service must provide in order to participate in collaboration. This interface can then be used during the development of the service in question. The choreography model can be used also to check (at design time) whether the behavioral interface of an existing service conforms to a choreography and thus, whether the service in question would be able to play a given role in that choreography.
Similarly, a behavioral interface can be used as a starting point to generate an *orchestration* skeleton that can then be filled up with details, regarding internal tasks, and refined into a full orchestration. On the other hand, an existing orchestration could be checked for consistency against an existing behavioral interface. In this way, it would be possible, for example, to detect

situations where a given orchestration does not send messages in the order in which these are expected by other services.
A more subtle dependency is semantic consistency of a global choreography and local process orchestrations. A choreography definition introduces message ordering constraints over the interface views of local process orchestration definitions. These need to be supported at the orchestration level in which they are mapped. The expressive power of orchestration semantics, at the same time, should not be limited by the choreography layer. Interactions supported by WS-CDL specifications occur between a pair of roles; in other words, only binary interactions are supported. Missing in WS-CDL is the explicit support for multi-party interactions and more complicated messaging constraints which these bring.

However, if we compare the development of WS-CDL with that of WSBPEL, we observe that WSBPEL stemmed from two sources, WSFL and XLang, which derived themselves from languages supported by existing tools (namely MQSeries Workflow and BizTalk). Furthermore, together with the first draft of WSBPEL, a prototype implementation was released. In contrast, WS-CDL has been developed without any prior implementation and does not derive (directly) from any language supported by an implementation (Dijkman, Dumas 2004). For this, we propose in this work to extend WSBPEL compositions with communications semantics and therefore both orchestration and choreography co-exist within one single environment.

COMMUNICATION SEMANTICS FOR BPEL PROCESSES

In our previous work (Rouached, Gaaloul and al. 2006; Rouached, Godart 2007), the design and implementation of Web service composition interactions was discussed and models were produced to provide a formal representation of the behaviour specified. These models are useful to describe individual compositions; however, an elaboration of modelling is required to represent the behaviour of interacting compositions across partnered processes. A series of compositions in Web service choreography needs specific modelling activities that are not explicitly derived from an implementation. In what follows, we describe this elaboration of models to support a view of interacting Web service compositions extending the mapping from WSBPEL to EC discussed in our previous work, and including Web service interfaces (WSDL) for use in modelling between services.

Motivating example

We present a brief overview of the problem along with our solution mechanism using the following running example implemented as a composition with several WSBPEL processes (see Figure 1). The collaboration represents a *loan approval* and includes three partners namely *credit approval, Risk Assesment and Decision*. Each of these components is implemented as a bpel process since it needs some other processes to ensure its role in the collaboration. A customer makes a new credit request to the *Credit Approval* composite service. The latter invokes the *CollectInfo* service to know more about the customer. Once the data are available, *Credit Approval* sends them to *Risk Assesment* composite service which considers whether the risk of the credit is low or high. The risk evaluation is then sent to the *Decide* composite service which decides even to approve the credit or not. If the assessment is low then the lowest rate is calculated and a reply is sent to the costumer following approval. If the assessment is high risk then the customer is asked to apply through an alternative process. Customer notification is achieved through the *Notify* service.

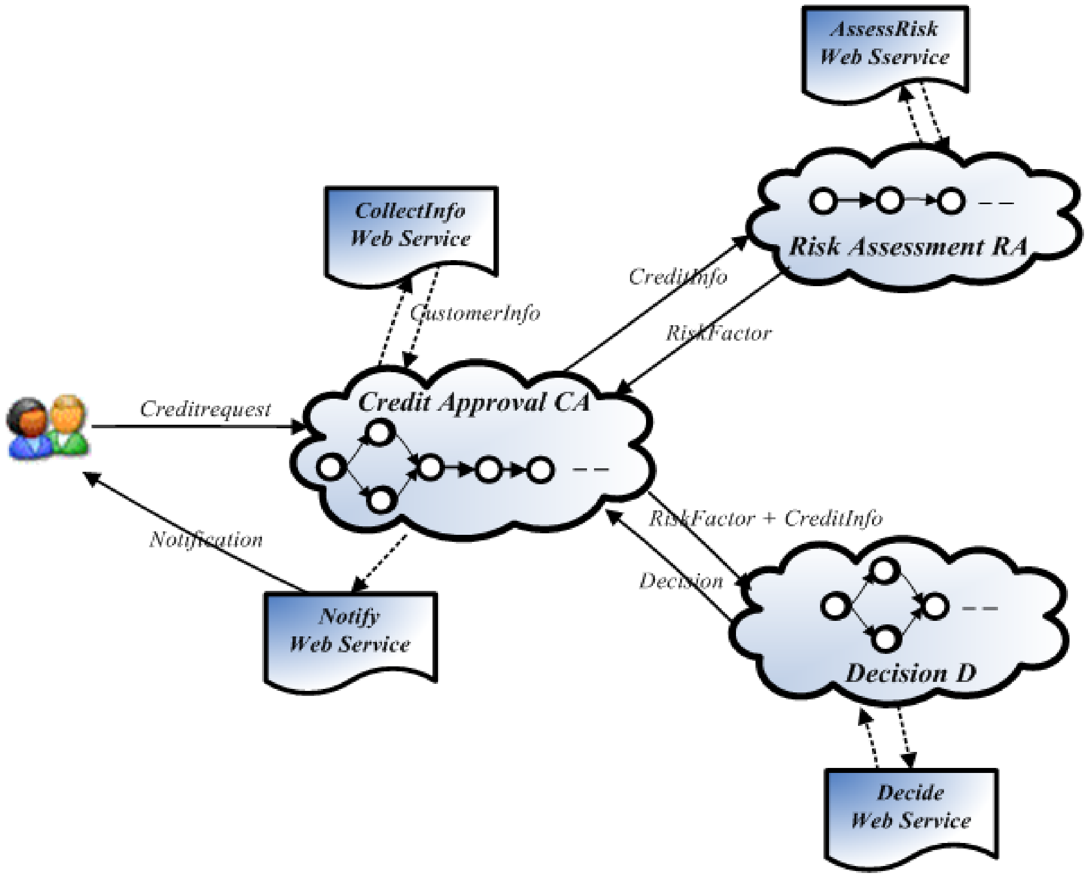


Figure 1. Credit approval process example

WSBPEL Language

WSBPEL introduces a stateful model of Web services interacting by exchanging sequences of messages between business partners. The major parts of a WSBPEL process definition consist of (1) the business process partners (Web services that the process interacts with), (2) a set of variables that keep the state of the process, and (3) an activity defining the logic behind the interactions between the process and its partners. Activities that can be performed are categorized into basic, structured, and scope-related activities. Basic activities perform simple operations like receive, reply, invoke and others. Structured activities impose an execution order on a collection of activities and can be nested. Then, scope-related activities enable defining logical units of work and delineating the reversible behavior of each unit. Below, we describe the main activities (basic and structured).

Basic Activities

Basic activities in a WSBPEL process support primitive functions (e.g. invocation of operations and assignments of variable values): (i) the invoke activity calls an operation in one of the partner services of the composition process, (ii) the receive activity makes the composition process to wait for the receipt of an invocation of its operations by some of its partner services, (iii) the reply activity makes the composition process to respond to a request for the execution of an operation previously accepted through a receive activity, (iv) the assign activity is used to copy the value from a variable to another one, (v) the throw activity is used to signal an internal fault, (vi) the wait activity is used to specify a delay in the process that must last for a certain period of time.

Structured Activities

Structured activities provide the control and data flow structures that enable the composition of basic activities into a business process: (i) the sequence activity includes an ordered list of other activities that must be performed sequentially in the exact order of their listing, (ii) the switch activity includes an ordered list of one or more conditional branches that include other activities and may be executed subject to the satisfiability of the conditions associated with them, (iii) the flow activity includes a set of two or more activities that should be executed concurrently. A flow activity completes when all these activities have completed, (iv) the pick activity makes a composition process to wait for different events (expressed by onMessage elements) and perform activities associated with each of these events as soon as it occurs, (v) the while activity is used to specify iterative occurrence of one or more activities as long some condition holds true.

Event Calculus

Event calculus (Kowalski, Sergot 1986) is a general logic programming treatment of time and change. The formulation of the event calculus is defined in first order predicate logic like the situation calculus. Likewise, there are actions and effected fluents. Fluents are changing their valuations according to effect axioms defined in the theory of the problem domain. However there are also big differences between both formalisms. The most important one is that in the event calculus, narratives and fluent valuations are relative to time points instead of successive situations. The most appearing advantage of this approach is the inherent support for concurrent events. Events occurring in overlapping time intervals can be deduced. Inertia is an assumption, which accounts a solution to the frame problem together with other techniques and it is saying that a fluent preserves its valuation unless an event specified to affect (directly or indirectly) the fluent occurs.

Each event calculus theory is composed of axioms. A fluent that holds since the time of the initial state can be described by the following axioms (Shanahan 1999):
$holdsAt(\mathrm{f}, t) \leftarrow initially(f) \wedge \neg clipped(t0, f, t)$
$holdsAt(\neg f, t) \leftarrow initially(\neg \mathrm{f}) \wedge \neg declipped(t0,f, t)$
Axioms below are used to deduce whether a fluent holds or not at a specific time.
$holdsAt(f, t) \leftarrow happens(e, t1, t2) \wedge initiates(e, f, t1) \wedge \neg clipped(t1, f, t) \wedge t2 < t$
$holdsAt(\neg f, t) \leftarrow happens(e, t1, t2) \wedge terminates(e, f, t1) \wedge \neg declipped(t1, f, t) \wedge t2 < t$
The predicate *clipped* defines a time frame for a fluent that is overlapping with the time frame of an event which terminates this fluent. Similarly *declipped* defines a time frame for a fluent which overlaps with the time frame of an event that initiates this fluent. The formula *initiates* (*e, f, t*) means that fluent *f* holds after event *e* at time *t*. The formula *terminates* (*e, f, t*) denotes that fluent *f* does not hold after event *e* at time *t*. The formula *happens* (*e, t1, t2*) indicates that event *e* starts at time *t1* and ends at time *t2*. The instantaneous events are described as *happens* (*e, t).*
$clipped(t1, f, t4) \leftrightarrow (\exists E,t2, t3) [happens(e, t2, t3) \wedge$
$terminates(e, f, t2) \wedge t1 < t3 \wedge t2 < t4]$
$declipped(t1, f, t4) \leftrightarrow (\exists e,t2, t3) [happens(f, t2, t3) \wedge$
$initiates(e, f, t2) \wedge t1 < t3 \wedge t2 < t4]$

Given the fact that we consider communications actions where ordering and timing are relevant and we adopt event driven reasoning, the event calculus 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.

Event driven specification

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

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)*

Table 2. Events expressed in EC

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, *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 in *Var1* 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.

Compositions interactions

Here we seek to further our modelling of Web service interactions through two viewpoints. Firstly, 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.

As mentioned so far, our objective 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.

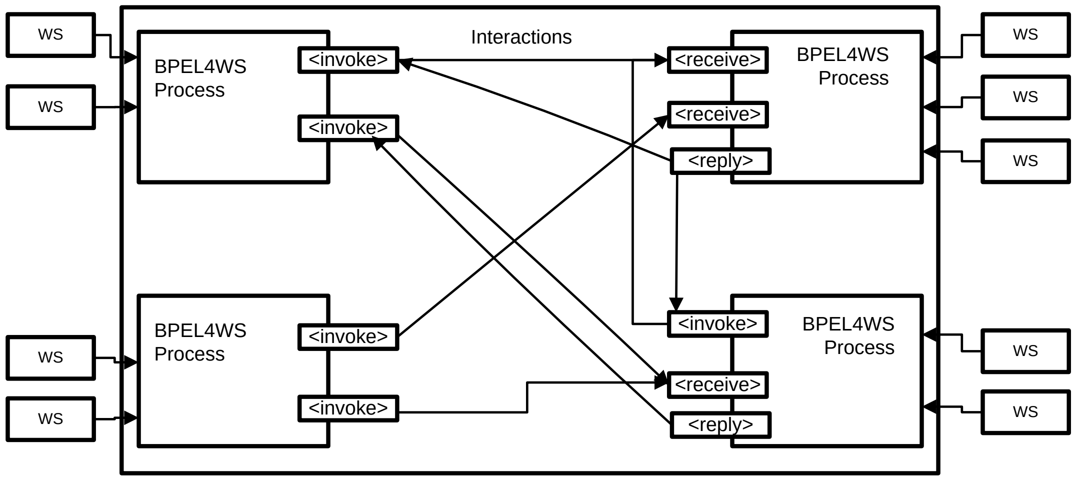


Figure 2. Web service compositions Interactions

To start, we require a process to analyze which activities are partnered in the composition. For example, *invoke* from the *Credit Aproval* process *CA* (a risk assessment request) will be received by the Risk Assessment process *RA* (*receive* a risk assessment request). Equally, the *CA invoke* activity, to check the decision for the credit request by contacting *Decision* process, will be aligned with *receive* in the *CA* 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 (Magee, Kramer 1999). 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, and (4) building interaction models using our formalism. We detail also, the interaction modelling algorithm we proposed.

Service conversations identification

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. It should be noted 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 (WSBPEL documents). The output of this phase is a list of interaction activities.

Service partners and roles identification

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 between a business process and 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.

Linking compositions 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*. The choreography modelling algorithm is shown in Algorithm 1 Where:

1- *O* is the set of all operations provided by a Web service in the choreography.
2- C_w is the WSBPEL 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: $In = \{w_i.o_n \mid O \leq n \leq n_{wi}\}$.
   - 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 $P = \{w_j.o \mid w_j \leq w\text{i} \text{ and } \exists j \in I, w_j.o \in In_j\}$
   - *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

```
begin
  For each composition C_wi do
    For each $a \in A_{wi}$ do
      $P\_local \leftarrow a.p$
      $P\_link \leftarrow P\_link.partnerLink$
      $PLT \leftarrow P\_link.partnerLinkType$
      $Port\_type \leftarrow PLT.portType$
      For each $In_{wj}$ ($w_j \in W_i$) do
        if $In_{wj}.porttype = Port\_type$ then
          $actual\_partner \leftarrow w_j$
          Lookup $w_j.o \in A_{wj}$ such as $w_j.o = a.o$

      if $a.o.out$ is in (rendez-vous style) then
        add invokeOutput action to activity model
        Build reply-invokeOutput connector

      Build invoke-receive connector

end
```

Algorithm 1. Choreography modelling 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.
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.

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 a 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.

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 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 (Fu, Bultan and al. 2004)).

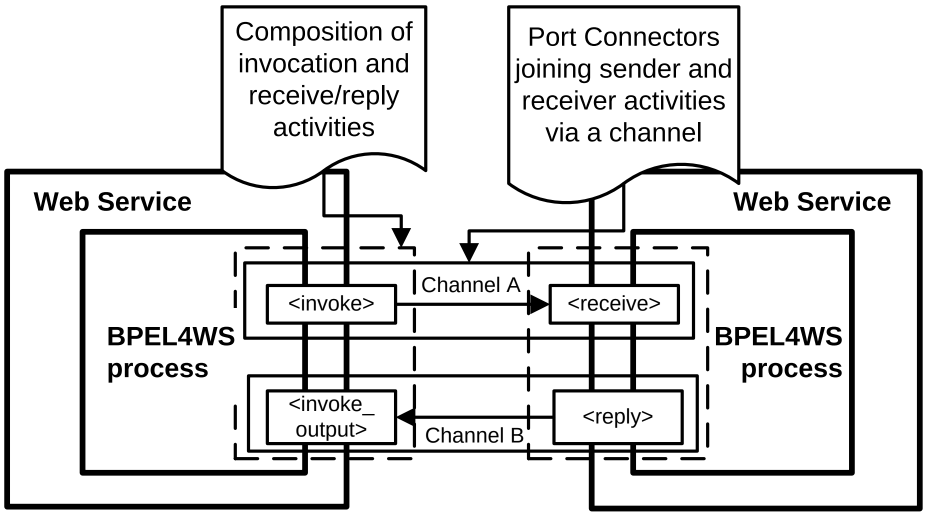

Composition of invocation and receive/reply activities
Port Connectors joining sender and receiver activities via a channel
Web Service
Web Service
BPEL4WS process
BPEL4WS process
<invoke>
Channel A
<receive>
<invoke_ output>
Channel B
<reply>

Figure 3. Channels and Interaction activities of Web service compositions

Request only invocation modelling (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 as follows.

∀(t1: time) happens (invoke-ic (PartnerService, Operation (oId, inVar)), t1)
=> (∃t2) happens (invoke-rc (PartnerService, Operation (oId)), t2) ∧
initiates (invoke-rc (PartnerService, Operation (oId)), equalTo (inVar1, inVar), t2)) ∧ (t1 < t2).

∀(t2: time) happens (invoke-rc (PartnerService, Operation (oId)), t2)) ∧
initiates (invoke-rc (PartnerService, Operation (oId)), equalTo (inVar1, inVar), t2)
=> (∃t1) happens (invoke-ic (PartnerService, Operation (oId, inVar)), t1) ∧ (t1 < t2).

Rendezvous style invocation modelling (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 this port model is represented as follows.

∀(t1: time) happens (invoke-ic (PartnerService, Operation (oId, inVar)), t1)
=> (∃t2) happens (invoke-rc (PartnerService, Operation (oId)), t2) ∧
initiates (invoke-rc (PartnerService, Operation (oId)), equalTo (inVar1, inVar), t2)) ∧ (t1 < t2).

∀(t2: time) happens (invoke-rc (PartnerService, Operation (oId)), t2)) ∧
initiates (invoke-rc (PartnerService, Operation (oId)), equalTo (inVar1, inVar), t2)
=> (∃t1) happens (invoke-ic (PartnerService, Operation (oId, inVar)), t1) ∧ (t1 < t2).

∀(t3: time) happens (reply (PartnerService, Operation (oId2, outVar)), t3)
=> (∃t4) happens (invoke-ir (PartnerService, Operation (oId2)), t4) ∧ initiates (invoke-ir (PartnerService, Operation (oId2)), equalTo (outVar1, outVar), t4)) ∧ (t3 < t4).

∀(t4: time) happens (invoke-ir (PartnerService, Operation (oId2)), t4)) ∧
initiates (invoke-ir (PartnerService, Operation (oId2)), equalTo (outVar1, outVar), t4)
=> (∃t3) happens (reply (PartnerService, Operation (oId2, outVar)), t3) ∧ (t3 < t4).

Considering the loan approval example introduced so far (figure 1), Figure 4 shows the corresponding model to an interaction scenario between the *Credit Approval CA* and *Risk Assessment RA* processes. The resulting model is interpreted as follows: The invocation event of the *CA* operation (*AssessRisk)* at time *t1* should be received by *RA* at time *t2* such as *t1<t2*. The response to this request should happen at time *t3* such as *t3>t2*, and be received by *CA* at time *t4>t3*.

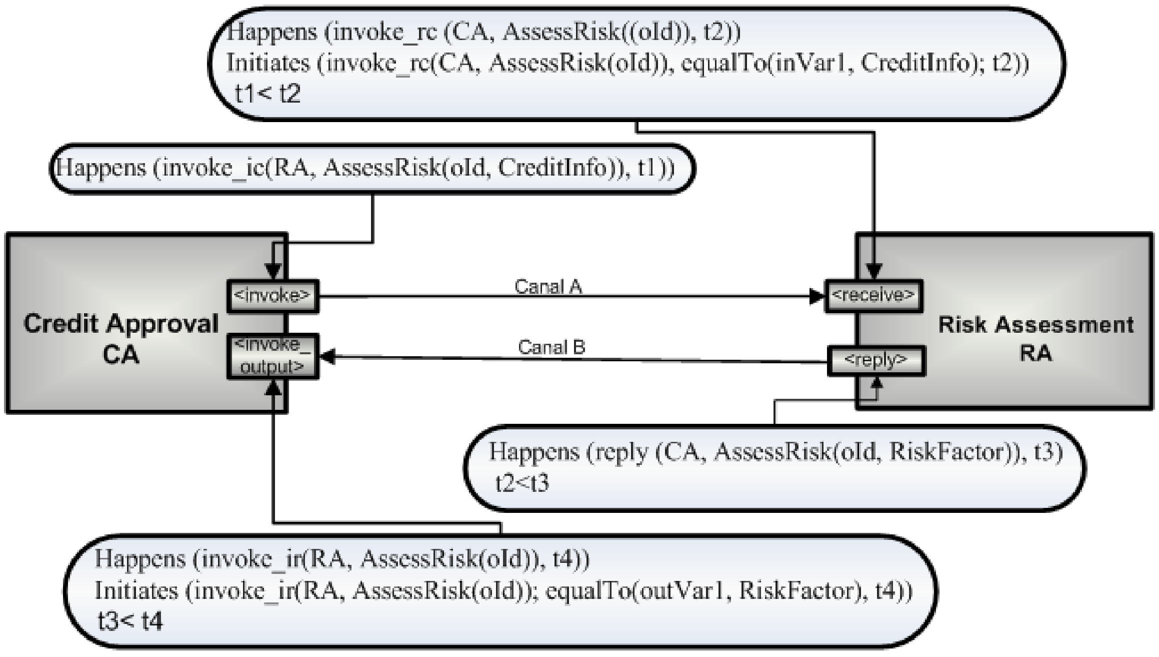

Happens (invoke_rc (CA, AssessRisk((oId)), t2))
Initiates (invoke_rc(CA, AssessRisk(oId)), equalTo(inVar1, CreditInfo); t2))
t1< t2
Happens (invoke_ic(RA, AssessRisk(oId, CreditInfo)), t1))
Credit Approval CA
<invoke>
<invoke_output>
Canal A
Canal B
<receive>
<reply>
Risk Assessment RA
Happens (reply (CA, AssessRisk(oId, RiskFactor)), t3)
t2<t3
Happens (invoke_ir(RA, AssessRisk(oId)), t4))
Initiates (invoke_ir(RA, AssessRisk(oId)); equalTo(outVar1, RiskFactor), t4))
t3< t4

Figure 4. Event invocation connectors

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 WSBPEL 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 2 lists the mapping explained here

WS interactions	Port action	BPEL actions (example)
Invoke (client)	Invoke-input	Invoke-CA-RA-AssessRisk
Receive (Partner)	Receive	Receive-CA-RA-AssessRisk
Reply (Partner to client)	Reply	Reply-RA-CA-AssessRisk
	Invoke-output	Output-RA-CA-AssessRisk

Table 2. Mapping process activities to port connectors

VERIFICATION AND VALIDATION

In the previous section we have described our approach to model Web service compositions with respect to their specification processes and interactions. These models provide a representation that can be used to perform verification and validation analysis using formal techniques. In this section we discuss this analysis.

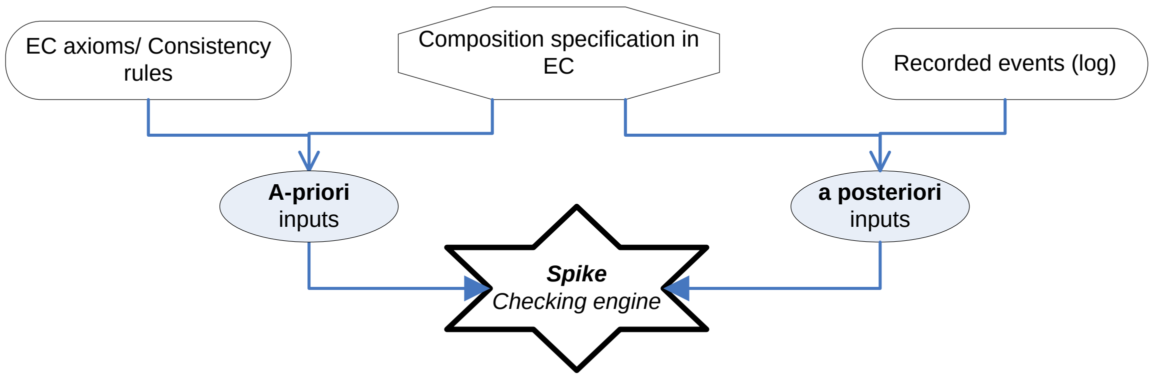


Figure 5. Verification overview

Recorded events

There are two main sources of data for Web log collecting, corresponding to the interacting two software systems: data on the Web server side and data on the client side. The existing techniques are commonly achieved by enabling the respective Web server's logging facilities. There already exist many investigations and proposals on Web server log and associated analysis techniques. Actually, papers on Web Usage Mining WUM (Punin, Krishnamoorthy et al. 2001) describe the most well-known means of Web log collection. Basically, server logs are either stored in the

Common Log Format[1] or the more recent *Combined Log Format*[2]. They consist primarily of various types of logs generated by the Web server. Most of the Web servers support as a default option the *Common Log Format*, which is a fairly basic form of Web server logging.

However, the emerging paradigm of Web services requires richer information in order to fully capture business interactions and customer electronic behavior in this new Web environment. Since the Web server log is derived from requests resulting from users accessing pages, it is not tailored to capture service composition or orchestration. That is why, we propose in the following a set of advanced logging techniques that allows to record the additional information to mine more advanced behavior.

Successful logging facilities for advanced architectures in Web Services models require composition (choreography/orchestration) information in the log record. Such information is not available in conventional Web server logs. Therefore, the advanced logging solutions must provide for both choreography or orchestration identifier and a case identifier in each interaction that is logged.
To adress this shortcoming, basically, we modify SOAP headers to include and gather the additional needed information capturing **choreography** details. Those data are stored in the special <*WSHeaders*>. This tag encapsulates headers attributes like: c*horeographyprotocol, choreographyname, choreographycase* and any other tag inserted by the service to record optional information; for example, the <*soapenv: choreographyprotocol*> tag, may be used to register that the service was called by WS-CDL choreography protocol. The SOAP message header may look as shown in Figure 6.

```
< soapenv : Header >
        < soapenv : choreographyprotocol
                soapenv : mustUnderstand = ”0”
                xsi : type = ”xsd : string” >WS−CDL
        < /soapenv : choreographyprotocol >
        < soapenv : choreographyname
                soapenv : mustUnderstand = ”0”
                xsi : type = ”xsd : string” > OTA
        < /soapenv : choreographyname >
        < soapenv : choreographycase
                soapenv : mustUnderstand = ”0”
                xsi : type = ”xsd : int” > 123
        < /soapenv : choreographycase >
< /soapenv : Header >
```

Figure 6. The SOAP message header

Concerning **orchestration** log collecting, since the most Web services orchestration are using a WSBPEL engine, which coordinates the various orchestration’s Web services, interprets and executes the grammar describing the control logic, we can extend this engine with a sniffer that captures orchestration information, i.e., the orchestration-ID and its instance-ID. This solution is centralized, but less constrained than the previous one which collects choreography information.

[1] http://httpd.apache.org/docs/logs.html
[2] http://www.w3.org/TR/WD-logfile.html

Finally, the focus is on collecting and analyzing **single** Web service composition instance. The exact structure of the Web logs or the event collector depends on the Web service execution engine that is used. In our experiments, we have used the engine bpws4j[3] that uses log4j[4] to generate logging events. Log4j is an Open Source logging API developed under the Jakarta Apache project. It provides a robust, reliable, fully configurable, easily extendible, and easy to implement framework for logging Java applications for debugging and monitoring purposes. The event collector (which is implemented as a remote log4j server) sets some log4j properties of the bpws4j engine to specify level of event reporting (INFO, DEBUG etc.), and the destination details of the logged events. At runtime bpws4j generates events according to the log4j properties set by the event collector. Figure 7 shows some example of log4j 'logging event' generated by bpws4j engine. The event extractor captures logging event and converts it to a unique log format. These expressions are described in next section.

```
2008-03-13 10:40:39,634       [Thread-35]     INFO   bpws.runtime – Outgoing response:
[WSIFResponse:serviceID = '{http://tempuri.org/services/CA}CustomerRegServicefb0b0-
fbc5965758--8000'operationName = 'completed'
          isFault = 'false' outgoingMessage = 'org.apache.wsif.base.WSIFDefaultMessage@
          1df3d59 name:null parts[0]:[JROMBoolean: : true]'
          faultMessage = 'null' contextMessage = 'null']
2008-03-13 10:40:39,634       [Thread-35]     DEBUG        bpws.runtime.bus -Response
          for external invoke is[WSIFResponse:serviceID='{http://tempuri.org/services
          /CA}CustomerRegServicefb0b0-fbc5965758--8000'
          operationName = 'authenticate'  isFault = 'false'         outgoingMessage =
          org.apache.wsif.base.WSIFDefaultMessage@1df3d59 name:null parts[0]:
          [JROMBoolean: : true]'faultMessage = 'null'     contextMessage = 'null']
2008-03-13 10:40:39,634       [Thread-35]     DEBUG        bpws.runtime.bus -Waiting
for request
```

Figure 7. Example of log4j 'logging event'

Our previous work (Rouached, Gaaloul and al. 2007) has contained more details and examples about the previous logging facilities.

Verification engine

As shown in Figure 5, the verification of the composition requirements can be done either *a-priori*, i.e., at design time, or *a-posteriori*, i.e., after runtime to test and repair design errors, and formally verify whether the process design does have certain desired properties.

The need for a-priori verification is important for compositions because they can be very complex processes, and therefore we need to check if the specified behavior is consistent, which is not a trivial task as soon as a composition process manages complex service dependencies. Indeed, these processes expect to enforce some high-level policies which we have defined in a set of consistency rules. Our interest is to use these rules specified formally in EC to check process consistency.

The a-posteriori verification is important to provide knowledge about the context of deviation and the reasons of discrepancies between process models and related instances. This kind of verification is necessary since some interactions between Web services that constitute a process

[3] http://alphaworks.ibm.com/tech/bpws4j

[4] http://logging.apache.org/log4j

may be dynamically specified at runtime, causing unpredictable interactions with other services, and making the a-priori verification method insufficient as it only takes into account static aspects.

Overview of SPIKE

Theorem provers have been applied to the formal development of software. They are based on logic-based specification languages and they provide support to the proof of correctness properties, expressed as logical formulas. In our work, we use the SPIKE induction prover (Stratulat 2001). SPIKE was chosen for the following reasons: (i) its high automation degree (to help a Web service designer), (ii) its ability on case analysis, (iii) its refutational completeness (to find counter-examples), and (iv) its incorporation of decision procedures (to automatically eliminate arithmetic tautologies produced during the proof attempt[5].

SPIKE proof method is based on cover set induction. Given a theory, SPIKE computes in a first step induction variable where to apply induction and induction terms which basically represent all possibles values that can be taken by the induction variables. Typically for a nonnegative integer variable, the induction terms are 0 and x+1, where x is a variable.
Given a conjecture to be checked, the prover selects induction variables according to the previous computation step, and substitutes them in all possible ways by induction terms. This operation generates several instances of the conjecture which are then simplified by rules, lemmas, and induction hypotheses.

Encoding EC in SPIKE

Here we describe a method for representing EC in SPIKE language. In the sequel, we assume that all formulas are universally quantified. Then the ingredients of this encoding are:

- ***Data***: All data information manipulated by the system are ranged over a set of sorts. These data concern generally the argument types of events and fluents. For instance, the sets of CustomerInfos, CreditInfos and RiskFactors are defined respectively by the sorts *Customer*, *Credit* and *Risk*. The sort *Bool* represents the Boolean values, where *true* and *false* are its constant constructors.
- ***Events***: We consider that all events of the system are of sort *Event*, where the event symbols are the constructors of this sort. These constructors are free as all event symbols are assumed distincts. For instance, the event symbol *Credit_request(x, y, z, t)* is a constructor of *Event* such that *x*, *y*, *z* and *t* are variables of sorts *Customer*, *CreditAproval, credit and Preference* respectively. We define also an idle event which when occuring it lets the system unchanged. We represent it by the constant constructor *Noact*.
- ***Fluents***: The sort *Fluent* respresents the set of fluents. All fluent symbols of the systems are the constructors of sort *Fluent*, which are also free. The fluent symbol *Less (risk1, riskmax)*, for example, means that the variables *risk1* and *riskmax*, of sort Risk, are such as risk1<riskmax.
- ***Time***: The sort of natural numbers, *Nat*, which is reflected by constructors *0* and successor *succ(x)* (meaning *x+1*).
- ***Axioms***: We express all predicates used in EC as Boolean function symbols. For instance *happens: Event * Nat -> Bool, initiates: Event *Fluent * Nat -> Bool, terminates : Event *Fluent * Nat ->Bool,* and *holdsAt : Fluent *Nat *Nat -> Bool* are the signatures of

[5] $x + z > y = \text{false} \wedge z + x < y = \text{false} \Rightarrow x + z = y$.

predicates *happens*, *initiates*, *terminates* and *holdsAt* respectively. Then, the EC axioms are expressed in conditional equations.

- ***Log***: Recorded logs are also expressed in equational form: *ListEvent= (e1, e2… en)*.
- ***Requirements***: In the same way, we express the composition requirements in equational form. For instance, a requirement that concerns a credit request can be represented by *happens(Credit_request(x,y,I,p),t1)=true* $\wedge$ *happens(AssessRisk(y,w,I,p),t2)=true =>* *(t1 < t2)=true*, where t1*, t2, x, y, w* , p and *i* are variables.

Finally, we build an algebraic specification from EC specification. Once building this specification, we can check all behavioural properties by means the powerful deductive techniques (rewriting and induction) provided by SPIKE.

Checking composition requirements

All the generated axioms can be directly given to the prover SPIKE, which automatically transforms these axioms into conditional rewrite rules. When SPIKE is called, either the requirement proof succeeds, or the SPIKE's proof-trace is used for extracting all scenarios which may lead to potential deviations. There are two possible scenarios. The first is meaningless because conjectures are valid but it comes from a failed proof attempt by SPIKE. Such cases can be overcome by simply introducing new lemmas. The second one concerns cases corresponding to real deviations. The trace of SPIKE gives all necessary informations (events, fluents and timepoints) to understand the inconsistency origin. Consequently, these informations help designer to detect behavioural problems in the composite Web service.

Below, we present a fragment of the SPIKE trace showing a deviation detection when checking a requirement (Figure 8).

```
Uncaught exception: Failure("fail induction on [ 10973 ] CreditInfo (u2, u1, u3, u5)
 <> Credit_request (e1, e2, e3, e4) /\\ CreditInfo (u2, u1, u3, u5) <> AssessRisk (e2, e5,
e3) /\\ u2 = e5 /\\ u1 = e2 /\\ u3 = e3 /\\ u5 = 3 /\\ u6 = 10 /\\ AssessRisk (u1, u2, u3) <>
Credit_request (e1, e2, e3, e4) /\\ u1 = e2 /\\ u2 = e5 /\\ u3 = e3 /\\ u4 = 3 => u6 < (u4 +
(6)) = true ;")while proving the following initial conjectures
[ 6584 ] Happens (p (AssessRisk (u1, u2, u3), u4)) = true /\ Happens (p (CreditInfo (u2,
u1, u3, u5), u6)) = true => u6 < (u4 + (6)) = true ;
Elapsed time: 0.186 s
We failed
```

Figure 8. A fragment of SPIKE trace example

Implementation

The validation tool that we have developed is shown in Figure 9. At run-time, a process execution engine executes the WSBPEL composition process and delivers the functionality of the process. This process execution engine is referred to as *composition execution environment*. *The composition manager* has responsibility for overseeing the monitoring of requirements regarding the composition process. *The BPEL2EC tool*, is built as a parser that can automatically transform a given WSBPEL process into EC formulas according to the transformation scheme (Rouached, Godart 2007). 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 behavioural specification of this composition in Event Calculus. The description of this implementation is beyond the scope of this paper and may be found in (Rouached, Godart 2007). Then, to support the choreography aspects introduced in this paper, we have extended the BPEL2EC tool with two xml parsers developed using the application programming interface JDOM (Java Document Object Model).

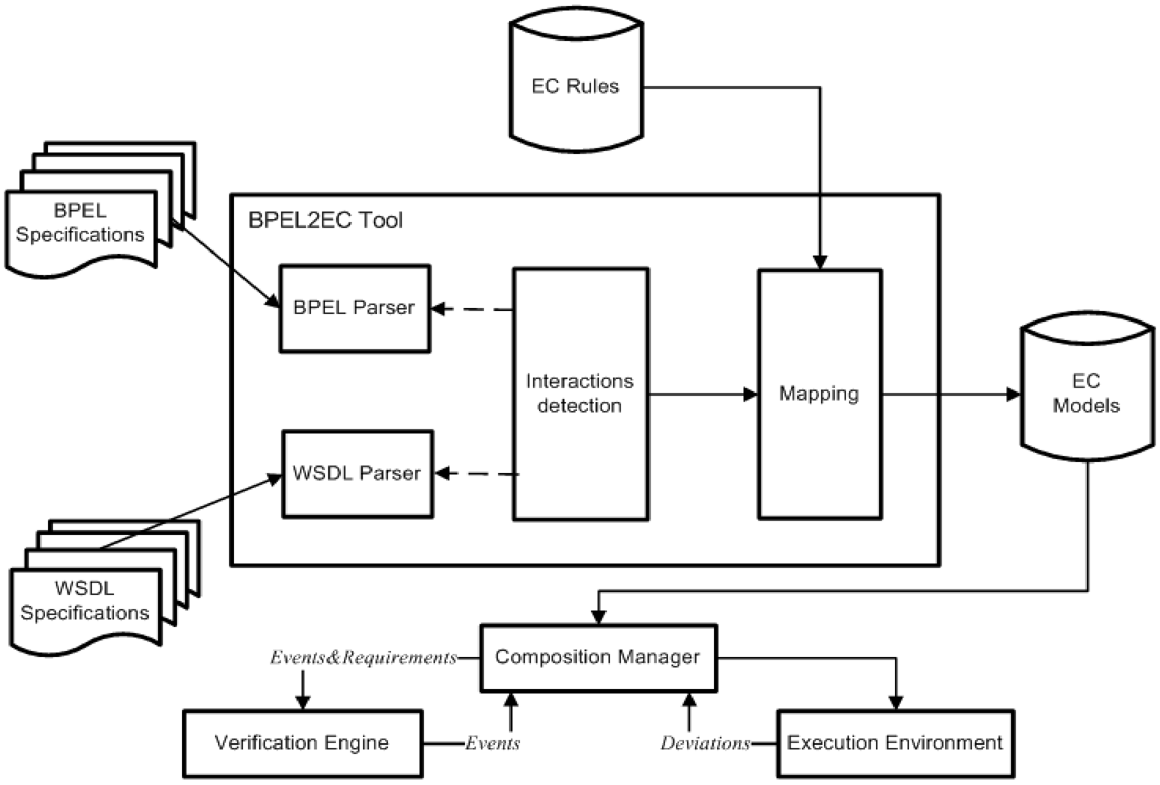
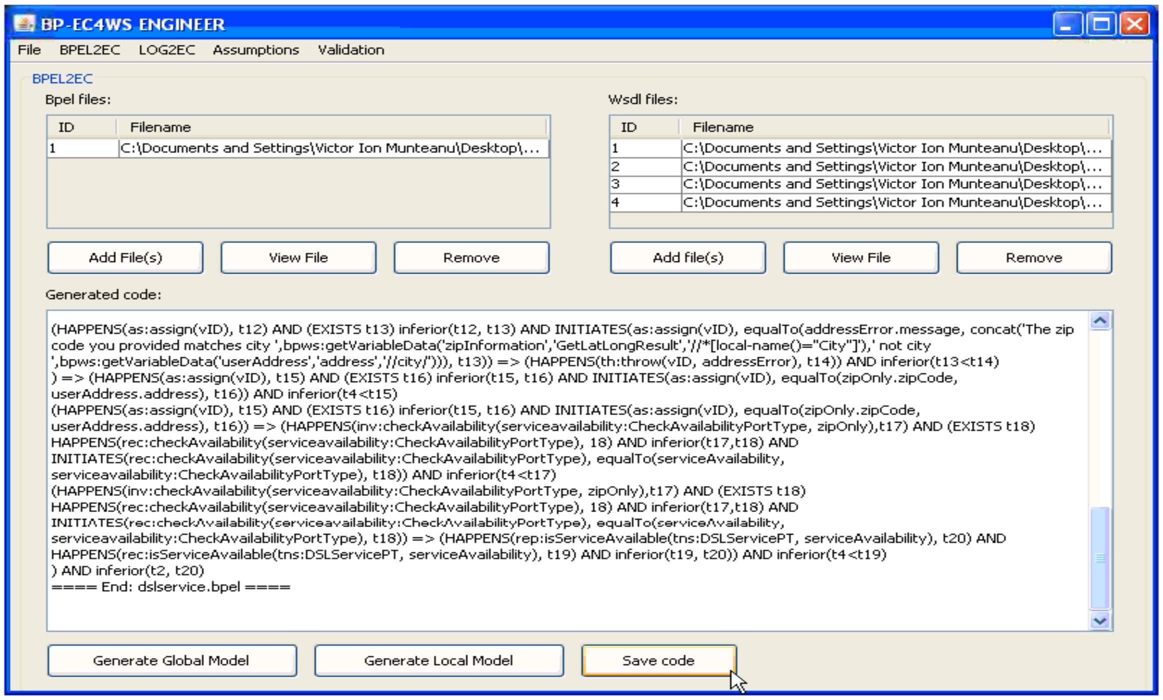


Figure 9. Validation Tool

The starting point is a set of Web service compositions 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 uses the EC translation rules defined so far in the paper 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 respect to the models. Figure 10 shows a snapshot of the validation tool in action. It shows how global models (choreography aspects) and local models (orchestration aspects) are generated in the same way. It gives also the possibility to save the resulting EC models to be used in the verification process.

The verification engine, shown in Figure 11 is responsible for checking requirements of the composition processes and their services at run-time. It consists of an EC checker that processes the events recorded in the event log and checks if they are compliant with the requirements of the composition. The check carried out determines whether the set of the recorded events generated by the composition process execution entails the negation of a requirement or not [6].



Bpel files:

ID	Filename
1	C:\Documents and Settings\Victor Ion Munteanu\Desktop\...

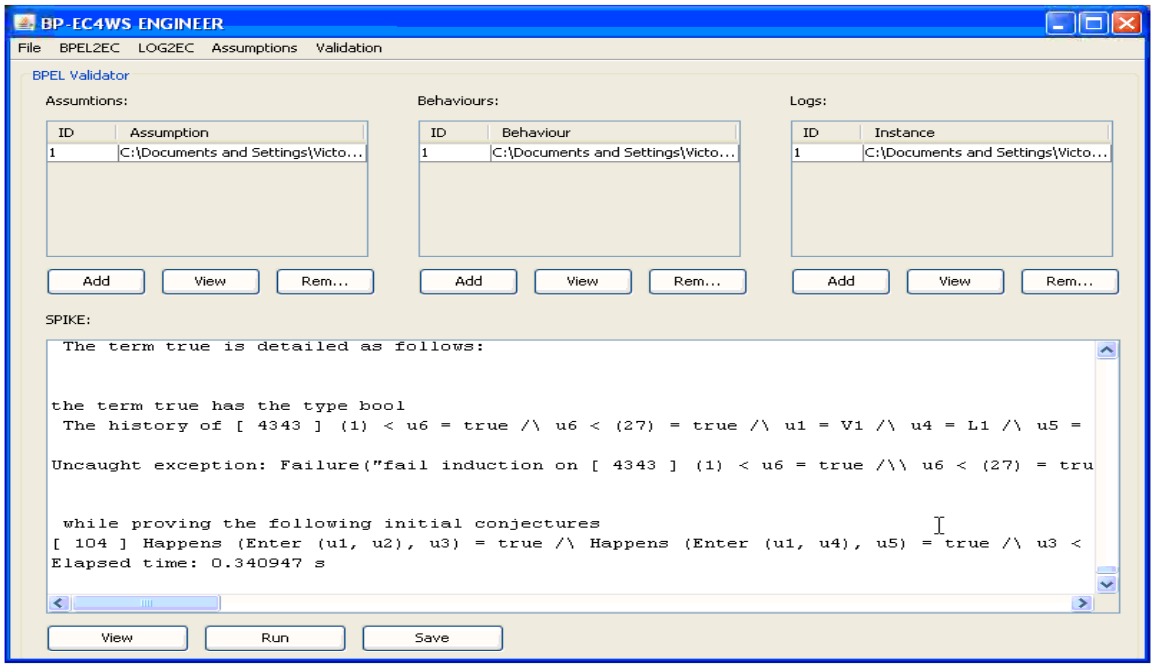
Wsdl files:

ID	Filename
1	C:\Documents and Settings\Victor Ion Munteanu\Desktop\...
2	C:\Documents and Settings\Victor Ion Munteanu\Desktop\...
3	C:\Documents and Settings\Victor Ion Munteanu\Desktop\...
4	C:\Documents and Settings\Victor Ion Munteanu\Desktop\...

Generated code:

```
(HAPPENS(as:assign(vID), t12) AND (EXISTS t13) inferior(t12, t13) AND INITIATES(as:assign(vID), equalTo(addressError.message, concat('The zip
code you provided matches city ',bpws:getVariableData('zipInformation','GetLatLongResult','//*[local-name()="City"]'),' not city
',bpws:getVariableData('userAddress','address','//city/'))), t13)) => (HAPPENS(th:throw(vID, addressError), t14)) AND inferior(t13<t14)
) => (HAPPENS(as:assign(vID), t15) AND (EXISTS t16) inferior(t15, t16) AND INITIATES(as:assign(vID), equalTo(zipOnly.zipCode,
userAddress.address), t16)) AND inferior(t4<t15)
(HAPPENS(as:assign(vID), t15) AND (EXISTS t16) inferior(t15, t16) AND INITIATES(as:assign(vID), equalTo(zipOnly.zipCode,
userAddress.address), t16)) => (HAPPENS(inv:checkAvailability(serviceavailability:CheckAvailabilityPortType, zipOnly),t17) AND (EXISTS t18)
HAPPENS(rec:checkAvailability(serviceavailability:CheckAvailabilityPortType), 18) AND inferior(t17,t18) AND
INITIATES(rec:checkAvailability(serviceavailability:CheckAvailabilityPortType), equalTo(serviceAvailability,
serviceavailability:CheckAvailabilityPortType), t18)) AND inferior(t4<t17)
(HAPPENS(inv:checkAvailability(serviceavailability:CheckAvailabilityPortType, zipOnly),t17) AND (EXISTS t18)
HAPPENS(rec:checkAvailability(serviceavailability:CheckAvailabilityPortType), 18) AND inferior(t17,t18) AND
INITIATES(rec:checkAvailability(serviceavailability:CheckAvailabilityPortType), equalTo(serviceAvailability,
serviceavailability:CheckAvailabilityPortType), t18)) => (HAPPENS(rep:isServiceAvailable(tns:DSLServicePT, serviceAvailability), t20) AND
HAPPENS(rec:isServiceAvailable(tns:DSLServicePT, serviceAvailability), t19) AND inferior(t19, t20)) AND inferior(t4<t19)
) AND inferior(t2, t20)
==== End: dslservice.bpel ====
```

Figure 10. A screenshot of the validation tool



Assumtions:

ID	Assumption
1	C:\Documents and Settings\Victo...

Behaviours:

ID	Behaviour
1	C:\Documents and Settings\Victo...

Logs:

ID	Instance
1	C:\Documents and Settings\Victo...

SPIKE:

```
 The term true is detailed as follows:


the term true has the type bool
 The history of [ 4343 ] (1) < u6 = true /\ u6 < (27) = true /\ u1 = V1 /\ u4 = L1 /\ u5 =

Uncaught exception: Failure("fail induction on [ 4343 ] (1) < u6 = true /\\ u6 < (27) = tru


 while proving the following initial conjectures
[ 104 ] Happens (Enter (u1, u2), u3) = true /\ Happens (Enter (u1, u4), u5) = true /\ u3 <
Elapsed time: 0.340947 s
```

Figure 11. A screenshot of the verification engine

[6] ¬ (∀s.Spec(s) => ¬R(s)) where *Spec(s)* is the specification of *s* and *R(s)* is the requirement about *s*.

RELATED WORK

Creating new services by combining a number of existing ones is becoming an attractive way of developing value added Web services. This pattern is not new but it does pose some new challenges which have yet to be addressed by current technologies and tools for Web service composition. WSBPEL (Arkin, Askary and al. 2004) 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. To that end, several methods for this purpose have been proposed. In particular, most researches conducted fall in the realm of AI planning. Despite all these efforts, the modelling and analysis of Web service composition still is a highly complex task. The complexity, in general, comes from the following sources. First, the number of services available over the Web increases dramatically during the recent years, and one can expect to have a huge Web service repository to be searched. Second, Web services can be created and updated on the fly, thus the composition system needs to detect the updating at runtime and the decision should be made based on the up to date information. Third, Web services can be developed by different organizations, which use different concept models to describe the services, however, there does not exist a unique language to define and evaluate the Web services in an identical means. Below, we present an overview of recent methods related to our work.

With respect to Web service analysis approaches, in particular BPEL processes, several works were described to capture the behavior of BPEL (Andrews, Curbera and al. 2003) in some formal way. Some advocate the use of finite state machines (Fisteus, Fernandez and al. 2004), others process algebras (Ferrara 2004), and yet others abstract state machines (Fahland, Reisig 2005) or Petri nets (Ouyang, Aalst and al. 2005; Martens 2005, Stahl 2004). 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 (Yang, Tan and al. 2005; Van Breugel, and Koshkina 2006). (Van Breugel and Koshkina 2006) 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.

There have been some works on providing formal semantics for Web service composition languages. In (Ankolekar, Burstein and al. 2002), the mark-up and semantics for DAML-S is described. They describe the notion of a “semantic Web” as a series of Web resources that provide services, which effect some action or change in the world, such as the sale of a product or the control of a physical device. The semantic Web should enable users to locate, select, employ, compose, and monitor Web-based services automatically. Whilst in (Duan, Bernstein and al. 2004), WSBPEL abstract processes are analyzed and semantics given on the construction of WSBPEL implementations behind this. WSBPEL and DAML-S are similar attempts at a standard for workflow of services. However, WSBPEL focuses more on business Web service orchestration whilst DAML-S is more generic in terms of any Web based service or object (Seeley 2003). In (Woodman, Palmer and al. 2004), the authors present an extension to the WSDL specification to describe the interactions between Web services. This, in turn, is mapped to π-calculus processes and sequencing formed using its operators. Tasks are represented as processes and dependencies linking the tasks, represented by channels (representing data dependencies in conditional linking). As WSBPEL extends WSDL with an abstract process this mapping is aimed more at the choreography level (where the inner process of a service is not directly observed).

In terms of choreography and Web service conversations, work on asynchronous Web service communication has been described in (Fu, Bultan and al. 2004; Fu 2004), with an example focus on the WSBPEL specification reported in (Fu, Bultan and al. 2004). 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 WSBPEL interaction modelling is achieved through the use of Guarded Finite State Automata (GFSA) which enables data dependencies to be modeled alongside process transitions. In (Brogi, Canal and al. 2004) the authors describe an approach to formalizing conversations, by way of mapping the WSCI standard to CCS for Web service choreography descriptions. The technique 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.

(Kazhamiakin, Pistore and al. 2006) describes an approach for the verification of Web service compositions defined by a set of WSBPEL 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, Calvanese and al. 2005a; Berardi, Calvanese and al. 2005b) 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, Brambilla and al. 2005) presents a high-level language and methodology for designing and deploying Web applications using Web services. In particular, the authors extend WebML (Ceri, Fraternali and al. 2000) 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.

In (Foster, Uchitel and al. 2004a; Foster, Uchitel and al. 2005a; Foster, Uchitel and al. 2005b) the authors have described the semantics of WSBPEL by way of mapping each of the WSBPEL constructs to the FSP algebra and building a model of the process behaviour. Then, they have described an elaboration of composition models to support a view of interacting Web service composition processes extending this mapping, 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 fulfil a choreography scenario and if the conversation protocol implement (by way of interaction sequences) is compatible with that of partnered services.

Amongst the assumptions in their semantic mappings of WSBPEL to FSP, they have considered that a process lifecycle begins at the first receive activity specified in the process document. The possibility of multiple start points as parts of a series of receive activities would affect the order in which activities are executed. Related to this is also a limitation on modelling the correlation attribute of activities, which are used to match returning or known clients to interact in long-running processes (in a message to correlation linking). They have not implemented a synchronisation of such events, but they anticipate these mappings would be evolved to consider this in our future work. The mapping does not consider translating event handling, as part of an activity scope. Such a mapping would however, take a form similar to the fault and compensation handling although the semantics behind event handling are much more towards a time based simulation basis.

Compared to our work, in contrast to FSP models, the EC ontology includes an explicit time structure that is independent of any (sequence of) events under consideration. This helps for managing cases where a number of input messages may occur simultaneously (risk of non-deterministic behavior). Second, the EC ontology is close enough to the WSBPEL specification to allow it to be mapped automatically into the logical representation. Thus, we use the same logical foundation for verification at both design time (static analysis) and runtime (dynamic analysis). Third, the semantics of non-functional requirements can be represented in EC, so that verification is once again straightforward. One other advantage of our work is that we provide a mechanism to check the models produced in our approach against trace runs output from WSBPEL process engine instances. This is one way to evaluate how accurate the translation is, although consequently, there is always the question of whether the engine itself has been built to standards. We can therefore only compare expected with actual results based upon an assumption that the implementation engine and execution of a process are on best endeavours.

Except the previous work, 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.

Another common pattern of the above attempts is that they adapt static verification techniques and therefore violations of requirements may not be detectable. This is because Web services that constitute a composition process may not be specified at a level of completeness that would allow the application of static verification, and some of these services may change dynamically at run-time causing unpredictable interactions with other services.

CONCLUSION AND FUTURE WORK

In this paper, we have described a modelling approach of a process defined for a multiple Web service compositions. We detailed an elaboration of models to support a view of interacting Web service compositions extending the mapping from WSBPEL to EC, and including Web service interfaces (WSDL) for use in modelling between services. To model conversations in the context of 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 uses as input partner service interfaces (in the form of a WSDL document) and the implementation of models created in the initial implementation synthesis. The output of the composition modelling is a list of composition mapping requirements and information on non-interaction activities encountered and unmatched partner process references. 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. The extension provides a representation that enables us to perform analysis of service interaction for behaviour properties. The approach to verifying and validating these properties has been also discussed.

The future opportunities from undertaking this work are as follows. The types of property used in verification are open to a much broader range than suggested in this work. Within this future work, we wish to continue describing behaviour by elaborating on the wider choreography aspects of partnered service compositions. This includes considering fault, compensation and transactional, security, privacy, and integrity within and between distributed processes. As part of this we are working on privacy and confidentiality policies in Web services compositions models.

REFERENCES

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller,D.,Smith, D., Thatte, S. ,Trickovic, I., and Weerawarana, S. (2003). Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, *International Business Machines Corporation, and Microsoft Corporation*, 2003.

Ankolekar, A., Burstein, M. and al. (2002). DAML-S: Web Service Description for the Semantic Web. 1st International Semantic Web Conference (ISWC), Sardinia, Italy.

Arkin, A., Askary, S., Bloch, B., and Curbera, F. (2004). Web services business process execution language version 2.0. *Technical report, OASIS*, December 2004.

Barros, A., Dumas, M., and Oaks, P. (2005). Critical overview of the Web services choreography description language (ws-cdl), March 2005. http://www.bptrends.com.

Berardi, D., Calvanese, D., Giacomo, G.D., Hull, R., Lenzerini, M., and Mecella, M. (2005a). Modeling data processes for service specifications in colombo. In M. Missiko and A. D. Nicola, editors, *EMOI-INTEROP, volume 160 of CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

Berardi, D., Calvanese, D., Giacomo, G.D., Hull, R., and Mecella, M. (2005b). 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.

Brogi, A., Canal, C., Pimentel, E., and Vallecillo, A. (2004). Formalizing Web service choreographies. Electr. Notes Theor. Comput. Sci., 105:73–94, 2004.

Ceri, S., Fraternali, P., and Bongio, A. (2000) Web modeling language (Webml): a modeling language for designing Web sites. Comput. Netw., 33(1-6):137–157, 2000.

Duan, Z., Bernstein, A. and al. (2004). Semantics Based Verification and Synthesis of WSBPEL Abstract Processes. 3rd IEEE International Conference on Web Services, San Diego, CA.

Fahland, D., and Reisig, W. (2005). ASM-based semantics for BPEL: The negative control flow. In D. Beauquier and E. B¨orger and A. Slissenko, editor, Proc. 12th International Workshop on Abstract State Machines, pages 131–151, Paris, France, March 2005.

Ferrara, A. (2004). 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.

Fisteus, J., Fernandez, L., and Kloos, C. (2004). Formal verification of WSBPEL 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.

Foster, H., Sebastian, U., Jeff, M., Jeff, K. (2003). Model-based Verification of Web Service Compositions, ase, pp.152, 18th IEEE International Conference on Automated Software Engineering (ASE'03), 2003

Foster, H., Uchitel, S., et al. (2004a). Compatibility for Web Service Choreography. 3rd IEEE International Conference on Web Services (ICWS), San Diego, CA, IEEE.

Foster, H., Uchitel, S., et al. (2005a). Tool Support for Model-Based Engineering of Web Service Compositions. 3rd IEEE International Conference on Web Services (ICWS2005), Orlando, FL, IEEE.

Foster, H., Uchitel, S., et al. (2005b). Using a Rigorous Approach for Engineering Web Service Compositions: A Case Study. 2nd IEEE International Conference on Services Computing (SCC2005), Orlando, FL, IEEE.

Fu, X. (2004). Formal Specification and Verification of Asynchronously Communicating Web Services. Phd Thesis, Santa Barbara, CA, USA,University of California, 2004.

Fu, X., Bultan, T., and Su, J. (2004). 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.

Kazhamiakin, R., Pistore, M., and Santuari, L. (2006). 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.

Kowalski, R., and Sergot, M.J. (1986). A logic-based calculus of events. New generation Computing 4(1), pages 67–95, 1986.

Magee, J., and Kramer, J. (1999). Concurrency: state models & Java programs. John Wiley & Sons, Inc., New York, NY, USA, 1999.

Manolescu, I., Brambilla, M., Ceri, S., Comai, S., and Fraternali, P. (2005). Model-driven design and deployment of service-enabled Web applications. ACM Trans. Inter. Tech., 5(3):439–479, 2005.

Martens, A. (2005). 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.

Ouyang, C., Aalst, W., Breutel, S., Dumas, M., and Verbeek, H. (2005). Formal Semantics and Analysis of Control Flow in WS-BPEL. BPM Center Report BPM-05-15, BPMcenter.org, 2005.

Pistore, M., Roveri, M., and Busetta, P. (2004). Requirements-driven verification of Web services. Electr. Notes Theor. Comput. Sci., 105:95–108, 2004.

Rouached, M., Gaaloul, G., van der Aalst, W., Bhiri, S., and Godart, C. (2006). 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.

Rouached, M., and Godart, C. (2007). 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.

Seeley, R. (2003). "Berners-Lee: Integrate Web services and Semantic Web. Quote from Gartner Web Services and Application Integration conference." From http://www.adtmag.com/article.asp?id=7662.

Shanahan M. P. (1999). The Event Calculus Explained. In Artificial Intelligence Today, Springer- Verlag Lecture Notes in Artificial Intelligence no. 1600, Springer-Verlag, pp. 409--430, 1999.

Stahl, C. (2004). Transformation von WSBPEL in Petrinetze (In German). Master’s thesis, Humboldt University, Berlin, Germany, 2004.

Stratulat, S. (2001). A general framework to build contextual cover set induction provers. *Journal of Symbolic Computation*, 32(4):403–445, 2001.

Van Breugel, F., and Koshkina, M. (2006). Models and verification of bpel. Available at http://www.cse.yorku.ca/ franck/research/drafts/tutorial.pdf, 2006.

Woodman, S., Palmer, D., et al. (2004). Notations for the Specification and Verification of Composite Web Services. 8th IEEE International Enterprise Distributed Object Computing (EDOC) Conference, Monterey, California.

Yang, Y., Tan, Q., and Xiao, Y. (2005). 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.

ABOUT THE AUTHOR

Dr. Mohsen Rouached is a postdoctoral researcher at Claude Bernard University of Lyon. He is a member of the BD team of the LIRIS research laboratory, France, where he is involved in the S-Cube European Network of excellence. Before joining LIRIS, he was a researcher in the ECOO team of the LORIA-INRIA research laboratory and teaching assistant at Nancy University. His research interests lie in the area of Service Oriented Computing, Business Process Management, Semantic Web and ontologies, Formal verification and validation techniques, and Process Intelligence.

Walid Fdhila is a phd student at Nancy University, France. He is a member of the ECOO team of the LORIA-INRIA research laboratory and teaching assistant at National Polytechnic Institute of Lorraine. His research interests are in the area of Decentralized Workflows, Service Oriented Computing and Business Process Management.

Prof. Dr. Claude Godart is full time Professor at Nancy University, France and scientific director of the INRIA ECOO project. His centre of interest concentrates on the consistency maintenance of the data mediating the cooperation between several partners. This encompasses advanced transaction models, user centric workflow and Web services composition models. He has been implicated in several transfer projects with industries (France, Europe, and Japan) for a wide range of applications including e-commerce, software processes and e-learning.