Analyzing Behavioral Compatibility for Web Service Choreography Using Colored Petri Nets and ASK-CTL

Maya Souilah Benabdelhafid
Constantine2 University
LIRE Laboratory
Chaab Essas
Constantine, Algeria
Email: mabenabdelhafid@gmail.com

Béatrice Bérard
Sorbonne University
LIP6 Laboratory
UPMC & CNRS
Paris, France
Email: beatrice.berard@lip6.fr

Mahmoud Boufaida
Constantine2 University
LIRE Laboratory
Chaab Essas
Constantine, Algeria
Email: mboufaida@umc.edu.dz

***Abstract*—Web services have become the technology of choice for Service-Oriented Computing (SOC) implementation. Their composition is a recent field that has seen a flurry of different approaches proposed towards the goal of flexible distributed heterogeneous inter-operation of software systems. These systems are usually derived from higher-level models rather than be coded at low level. In practice, achieving Web service compatibility nonetheless continues to require significant efforts for modeling at multiple abstraction levels. Existing formal approaches typically require the analysis of the global space of joint executions of interacting Web services. We propose a formal approach where Web service choreography is represented with the high-level model of Colored Petri Nets (CPNs). ASK-Computational Tree Logic (ASK-CTL) is used to describe the behavioral compatibility of these services in terms of message order properties. Then, model checking is applied for the verification of these properties. The effectiveness of our work has been validated with the recent version of CPN Tools.**

***Keywords*-Web Service Choreography; Behavioral Compatibility; Model Checking; CPN; ASK-CTL.**

I. INTRODUCTION

SOC is a new computing paradigm that utilizes services as the basic constructs to support the development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments [1]. Web services [2] are considered as one of the most promising computing paradigms, which work as plugin mode to provide the value-added applications in SOC and Service-Oriented Architecture (SOA) [3]. They may use the Internet as the communication medium and open Internet-based standards, such as the Simple Object Access Protocol (SOAP) as transmission medium and the Web Services Description Language (WSDL) for their description. They currently support the externalization of atomic business capabilities [4]. Specifically, it is commonly accepted that a Web service description should include not only the interface, but also the business protocol supported by the service (i.e., its behavior, which is the specification of possible message exchange sequences that it supports). Services can be composed through choreography and orchestration. Choreography describes the interactions between participating services to the business process from a global perspective, while orchestration uses a central coordinator. Many composition methods as well as several proposals, such as Web Services Business Process Execution Language (WSBPEL) [2] for orchestration or Web Service Choreography Definition Language (WSCDL) [5] for choreography, have been brought forward to construct and describe the interactions among services. However, they are concerned only with syntactic or semantic compatibility among services, and the behavioral compatibility is ignored.

Behavioral compatibility analysis for Web service composition is one of the most important topics. In this paper, our goal is to investigate this topic in the context of choreography. We provide a formal basis for developing demonstrably correct choreography. Our definition for this correctness is related to message order requirements. We consider the problem of choreographing Web services from a high-level, conceptual perspective, that abstracts from the details of the interaction paradigm. As pointed by De Backer et al. [6], the first step of verifying if two Web services are compatible should occur on an abstract level that hides unnecessary underlying coordination and allows to focus on high-level units of collaboration. This simplifies the verification and provides a first step towards a compatibility before investigating details of a Web service description such as the content of a message. We propose the modeling of Web services and their choreography using CPNs [7] and show how a model checking technique can be employed to verify if the modeled choreography satisfies the order properties given as ASK-CTL [8] formulas. The CPN models are implemented using the recent version of the software CPN Tools (CPN Tools 4.0 [9]). The ASK-CTL toolkit provided with this tool is used to perform automated verification in order to prove that a service choreography is correct at design time. This is an important step towards reliable service choreography composition, since problems could be detected early in the development cycle, before even starting the implementation.

The rest of this paper is structured as follows. In Section II, we give a simple illustrating example of Web service behaviors in a choreography. Formal definitions of CPNs and ASK-CTL are recalled in Section III, with a brief description of the model checking technique. The formalization of behavioral

 ISBN: 978-1-61208-337-7

compatibility is presented in Section IV. Related works are discussed in Section V. Conclusions and future works are presented in Section VI.

II. Motivating Example

Let us consider a simple example where the scenario is that of a travel agency, with the cooperation of four partners:

1) *Travel Agency* has two main tasks: airline booking and hotel reservations,
2) *Bank* acts as a financial intermediary between the Airline company (respectively the Hotel) and the Travel Agency,
3) *Airline Company* sells flight tickets to Travel Agencies,
4) *Hotel* proposes nights to Travel Agencies.

The last three partners want to provide functionalities to the Travel Agency partner using the Web service technology. Each partner is a published Web service, participating in a choreography and is modeled as a business process including the description of its partners (or a link permitting to get it), the description of its interface (but not its local operations), and the description of an abstract process that represents its behavior (exchanged messages). The behavior of the four Web services is as follows (see Fig. 1.).

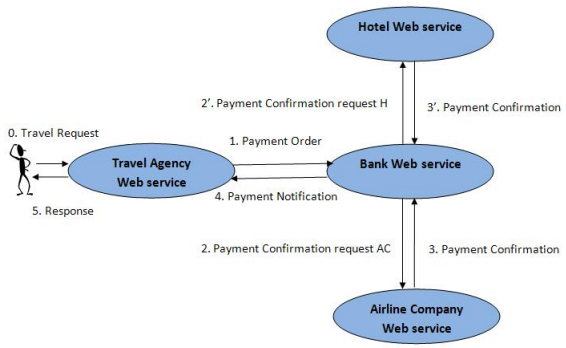

Fig. 1. A Web service choreography: A Travel Agency Example

First, a customer contacts the Travel Agency Web service and chooses its travel plan including information about the order and the payment method. Consequently, this service contacts the Bank Web service to pay the Airline Company (respectively the Hotel) Web service. Next, the Bank pays the Airline Company (respectively the Hotel) and asks them for the payment confirmation. The Airline Company (respectively the Hotel) sends its confirmation. If the payment operations are completed successfully then the Travel Agency contacts its customer and confirms his travel plan, and if one of them fails then it contacts the customer and asks if any other plan suits him or to cancel his request. The messages exchanged between the four Web services have constraints of order forming their behaviors.

The possible scenarios can be the following message ordering sequences: $0, 1, 2, 3, 2', 3', 4, 5$ or $0, 1, 2, 2', 3, 3', 4, 5$ or $0, 1, 2, 2', 3', 3, 4, 5$ or $0, 1, 2', 3', 2, 3, 4, 5$ or $0, 1, 2', 2, 3', 3, 4, 5$ or $0, 1, 2', 2, 3, 3', 4, 5$.

To guarantee the successful execution of these scenarios, Web services need to be verified formally in order to ensure that mutual interactions between them do not lead to any conflict. Specifically, we need to verify their compatibility. There are three aspects of service compatibility: syntactic, semantic, and behavioral [10]. Syntactic compatibility means that the structural interfaces of the interacting services are consistent. Semantic compatibility means that the interacting services exchange information that can be understood in a consistent and unambiguous way. Finally, behavioral compatibility means that the interacting services agree on what to expect from each other in terms of operations to execute, outcomes to deliver, and messages to be sent and received.

The static compatibility including the syntactic and semantic compatibility is essential to be checked. Checking the behavioral one, however, is a much more challenging task. In the example, the four partners may be syntactically and semantically compatible in interfaces, but they can behave improperly for the message exchange protocol. An example of behavioral property that we will later check is the following requirement: *The payment confirmation will be sent by the Airline Company after it receives the payment confirmation request.* It is obvious that if this property is not satisfied, then the collaboration leads to an erroneous message ordering even if they are syntactically and semantically consistent. Thus, the behavior of services must be taken into account in composition. The manual checking of service compatibility would clearly be error-prone and time consuming. Consequently, an approach to realize automatic and transparent checking is necessary.

This example will be modeled and the above behavioral property will be verified in order to respect the six anticipated scenarios.

III. Background

In this section, we briefly recall some formal definitions related to model checking formulas of the ASK-CTL logic for CPNs.

A. Colored Petri Nets

CPNs represent today one of the most widely used formalism incorporating data and hierarchy [11]. They are a discrete-event modeling language combining PNs and the functional programming language CPN ML. Initially, CPNs were supported by Design/CPN, later replaced by CPN Tools that supports the design of complex processes and the analysis of such processes using simulation and state space analysis.

In this section, we first recall definitions of CPNs that will be useful in establishing a CPN model for Web service choreography. These definitions are presented here in a simple way in order to adapt them to our problem of behavioral compatibility. A relation between them and CPN Tools 4.0 notations is also presented.

Definition 1 (Multi-set)**.** *A multi-set over a non-empty set Z is a mapping $b : Z \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers. The support of b is the set $supp(b) = \{z \in Z \mid b(z) \neq 0\}$. We denote by $Bag(Z)$ the set of multi-sets over Z with*

finite support and we write sometimes explicitly $b \in Bag(Z)$ *as* $b = \sum_{z \in Z} b(z)z$. *An* order relation, *an* addition *and a* difference *on multi-sets are defined as follows. For two multi-sets* $b, b' \in Bag(Z)$*:*

- $b \leq b'$ *if for all* $z \in Z$, $b(z) \leq b'(z)$,
- $b + b'$ *is given by* $\sum_{z \in Z}(b(z) + b'(z))z$.
- *if* $b \leq b'$, *then* $b' - b = \sum_{z \in Z}(b'(z) - b(z))z$.

Remark 1. In CPN Tools 4.0, special symbols are used for multi-sets: the order relation is noted $\ll$, the addition is noted $++$, with $\overset{++}{MS}\sum$ for a sum, and the symbol ‘ is placed between $b(z)$ (the multiplicity) and z (the element). These notations appear in Figures 3 and 4 for instance, where CPNs are extracted from the graphical interface of the tool.

CPN definitions use a set Σ of color domains containing the set $Bool = \{true, false\}$ and a set V of variables. The variables are typed by the function $Type : V \to \Sigma$ and we consider a set $Exp(V)$ of expressions using elements of V as free variables (or the empty expression).

Definition 2 (CPN Syntax)**.** *A CPN over the set of color domains* Σ *and the set of variables* V *is a 5-uplet* $N = (P, T, C, E, M_0)$ *where:*

- P *is a finite set of* places,
- T *is a finite set of* transitions *such that* $P \cap T = \emptyset$,
- $C : P \to \Sigma$ *associates a* color domain *with each place,*
- $E : P \times T \cup T \times P \to Exp(V)$ *associates with each pair* (p, t) *or* (t, p) *an* expression *typed as a multi-set over the color domain of the place:* $Type(E(p, t)) = Bag(C(p))$ *and* $Type(E(t, p)) = Bag(C(p))$,
- M_0 *is the* initial marking, *with* $M_0(p) \in Bag(C(p))$ *for each place* $p \in P$.

Remark 2. We do not define explicitly the set of arcs to simplify the notations and we use the habitual convention of Petri nets: the expression is empty if there is no arc, an empty expression evaluating to an empty multi-set.

The following definition presents the semantics of CPNs.

Definition 3 (CPN semantics)**.** *The semantics of a CPN* N *is described by a transition system* $\mathcal{T}_N = (\mathcal{M}, M_0, \to)$*:*

- *the configurations of* $\mathcal{M}$ *are markings* M, *with* $M(p) \in Bag(C(p))$ *for each place* $p \in P$,
- *the initial configuration is the initial marking* M_0,
- *the* transition relation $\to$ *is defined as follows.*
  Let t *be a transition and let* v *be a valuation of variables. We write* $v^-(p, t) \in Bag(C(p))$ *and* $v^+(t, p) \in Bag(C(p))$ *for the respective values of* $E(p, t)$ *and* $E(t, p)$ *for* $p \in P$.
  The transition $M \xrightarrow{t,v} M'$ *is possible if, for each place* $p \in P$, $M(p) \geq v^-(p, t)$ *and in this case,*
  $M'(p) = M(p) - v^-(p, t) + v^+(t, p)$ *for each* $p \in P$.

An execution *starting from* M *is a sequence of firings* $M \xrightarrow{t_1,v_1} M_1 \xrightarrow{t_2,v_2} M_2 \ldots$. *A marking* M' *is reachable from* M *if there exists a finite execution* $M \xrightarrow{t_1,v_1} M_1 \xrightarrow{t_2,v_2} M_2 \ldots \xrightarrow{t_n,v_n} M_n$ *starting from* M *such that* $M' = M_n$.

Remark 3. In this definition, a single transition is fired to avoid the steps in the presentation of Jensen [7]. This is not a problem because a step can be represented by the successive firing of several transitions.

B. The logic ASK-CTL

The logic ASK-CTL of CPN Tools (see [12] for more details) is an extension of the standard CTL [13]. An ASK-CTL formula is interpreted over the transition system $\mathcal{T}_N$ (called State Space (SS) in the tool) associated with a CPN model N and takes into account both configuration information (on markings, also called states) and transition information, thus extending CTL, where only configurations are labeled with sets of atomic propositions. The model checker of CPN Tools checks if such a formula holds over $\mathcal{T}_N$.

In the following definition, we consider the transition system $\mathcal{T}_N$ of a given CPN N. Operators $\neg, \wedge$ are boolean negation and conjunction, $\langle . \rangle$ is an existential ”next” modality, U is the standard until modality of CTL and E, A are respectively the existential and universal quantifiers on executions from CTL.

Definition 4 (ASK-CTL Syntax)**.** *The ASK-CTL logic has two categories of formulas: state and transition formulas, defined by mutual induction.*

State formulas are given by the grammar:
$\mathcal{A} ::= \alpha | \ \neg\mathcal{A} \ | \mathcal{A}_1 \wedge \mathcal{A}_2 | \ \mathsf{EU}(\mathcal{A}_1, \mathcal{A}_2) | \ \mathsf{AU}(\mathcal{A}_1, \mathcal{A}_2) | \ \langle\mathcal{B}\rangle$
where α *is a mapping from the set* $\mathcal{M}$ *of markings into booleans,* $\mathcal{A}, \mathcal{A}_1, \mathcal{A}_2$ *are state formulas and* $\mathcal{B}$ *is a transition formula.*

Transition formulas are given by the grammar:
$\mathcal{B} ::= \beta | \ \neg\beta \ | \beta_1 \wedge \beta_2 | \ \mathsf{EU}(\beta_1, \beta_2) | \ \mathsf{AU}(\beta_1, \beta_2) | \ \langle\mathcal{A}\rangle$
where β, β_1, β_2 *are mappings from the set of pairs* (t, v) *labelling transitions into booleans and* $\mathcal{A}$ *is a state formula.*

The semantics of ASK-CTL is defined inductively on configurations of the transition system $\mathcal{T}_N$ in the spirit of CTL, from the basis case: A configuration M satisfies α, written $M \models \alpha$, if $\alpha(M)$ is true. For instance:
- $M \models \mathsf{EU}(\mathcal{A}_1, \mathcal{A}_2)$ if there exists an execution $M \xrightarrow{t_1,v_1} M_1 \xrightarrow{t_2,v_2} M_2 \ldots \xrightarrow{t_n,v_n} M_n$ starting from M such that M_n satisfies $\mathcal{A}_2$ and all markings from M to M_{n-1} satisfy $\mathcal{A}_1$.
- $M \models \mathsf{AU}(\mathcal{A}_1, \mathcal{A}_2)$ if for all executions starting from M, there exists a marking M' satisfying $\mathcal{A}_2$ with all intermediate markings satisfying $\mathcal{A}_1$.

The next modality is similar to the one from the μ-calculus: $M \models \langle\mathcal{B}\rangle$ if there is a transition $M \xrightarrow{t,v} M'$ from M satisfying $\mathcal{B}$, as defined below.

The semantics of transition formulas is defined similarly:
- A transition $e = M \xrightarrow{t,v} M'$ satisfies β if $\beta(t, v)$ is true.
- $e \models \langle\mathcal{A}\rangle$ if M' satisfies $\mathcal{A}$.
- The formulas $\mathsf{EU}(\beta_1, \beta_2)$ and $\mathsf{AU}(\beta_1, \beta_2)$ are then defined like above on executions starting by e, with β_1 and β_2 satisfied by successive transitions instead of configurations.

Note that we may also use the standard abbreviations $\mathit{false} = \alpha \wedge \neg\alpha$, $\mathit{true} = \neg\mathit{false}$, $\varphi \to \psi = \psi \vee \neg\varphi$, $\mathsf{AF}\varphi = \mathsf{AU}(true, \varphi)$ and $\mathsf{AG}\varphi = \neg\mathsf{AF}(\neg\varphi)$.

 ISBN: 978-1-61208-337-7

C. Model Checking

A model checking procedure answers the following question: Given a state ASK-CTL formula $\mathcal{A}$ and a CPN N, does the initial configuration M_0 of $\mathcal{T}_N$ satisfy $\mathcal{A}$? For this, it must be able to answer any similar question on reachable configurations or transitions of $\mathcal{T}_N$. The tool uses Standard ML (SML) functions for this purpose. For instance, checking a state formula is expressed in SML by a function $eval\_node :< formula >, < node >$, which takes two arguments: the formula to be checked and a configuration (called node in the tool) from where the model checking should start. The mappings α are defined in SML by functions like $\mathsf{NF}(< message >, < node\ function >)$, where $node\ function$ takes a node and returns a boolean and $message$ is used when a formula evaluates to false. Similarly, the mappings β are defined by functions like $\mathsf{AF}(< message >, < arc\ function >)$. A formula $\mathsf{EU}(\alpha_1, \alpha_2)$ for two mappings α_1 and α_2, simply translates in SML as $\mathsf{EXIST\_UNTIL}(\alpha_1, \alpha_2)$, and so on.

Now, we deal with our proposed formal approach.

IV. A CPN AND ASK-CTL -BASED APPROACH

Our approach analyzes behavioral compatibility using the above definitions of CPNs and ASK-CTL. It is composed of two related phases (see Fig. 2.):

- *Choreography Modeling and Validation*: Modeling a Web service choreography by constructing Web service behaviors based on CPNs semantics and composing them. This modeling is validated by multiple simulations using CPN Tools 4.0. The result is a behavioral model to check.
- *Behavioral Properties Checking*: Verifying some behavioral properties on the generated behavioral model in terms of message order using the model checking technique described above. We first formally describe the behavioral properties as ASK-CTL formulas. Subsequently, we rewrite these formulas into SML format. In this way, a concrete formalization of the behavioral properties is obtained. The verification of these properties will be done over the transition system (or *SS*) that has been generated from the behavioral model by CPN Tools 4.0.

A. Choreography Modeling and Validation

In this first phase, we have three related steps: *CPN Modeling, Simulation, and CPNs Composition*.

1) CPN Modeling: According to the Web service behaviors, which are specified by the informal language Unified Modeling Language Diagram Activities (UML DA [14]), we construct a formal model for each Web service behavior based on CPN semantics such as the choreography may require different instances of a participating Web service. Consequently:

- the behavior execution states are captured by places.
- the message type (Web service instances and its incoming messages) is captured by the color set of the token (we do not look into the content of a message as it is not known until run time).
- the operation of its instance is captured by a transition (send or receive).

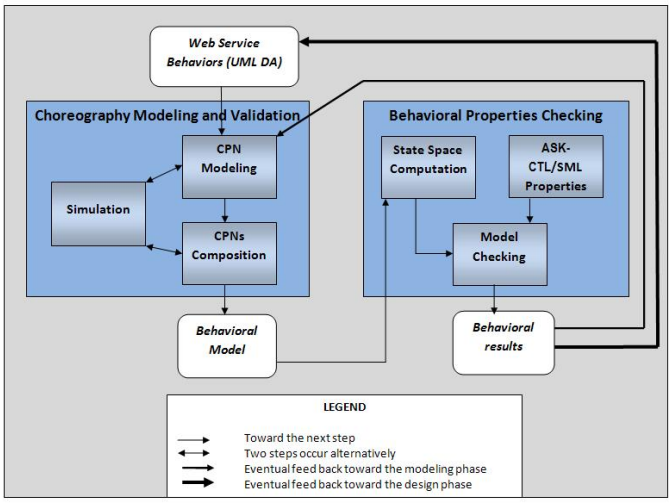

Fig. 2. Overview of our proposed approach

- the Web service initial state is captured by the initial marking M_0.

In our modeling, a Web service behavior is a conversation protocol that is defined as a CPN N where:

- the set of colors is $\Sigma = \{INS, I, MSGSTATE, I \times MSGSTATE, INS \times I \times MSGSTATE\}$, where:
  - INS is a color set, which defines the Web service instances:
    $colset\ INS = with\ ins1|ins2;$
    We define a variable x having as type INS:
    $var\ x : INS;$
  - the static subclasses of $I \times MSGSTATE$ include I, which is an integer type that represents the message identifier and $MSGSTATE$, which is an enumeration type that represents a message:
    $colset\ I = int;$
    $colset\ MSGSTATE = with\ TravelRequest$
    $|Response|PaymentOrder|PaymentNotification$
    $|PaymentConfirmationRequestAC$
    $|PaymentConfirmationRequestH$
    $|PaymentConfirmation;$
    we define two variables $msgid$ and $currentstate$ and $m0, m1, m4, m5$ having respectively as type I, $MSGSTATE$, and $I \times MSGSTATE$:
    $var\ msgid : I;$
    $var\ currentstate : MSGSTATE;$
    $var\ m0, m1, m4, m5 : I \times MSGSTATE;$
  - we define two variables $xm2, xm3$ having as type $INS \times I \times MSGSTATE$:
    $var\ xm2, xm3 : INS \times I \times MSGSTATE;$
  - we define also functions that will be attached to transitions and eventually two arcs, for example the function $startSend1$ that allows the sending of the message $m1$ that represents the Payment Order in Fig. 1. It is defined as follows:
    $fun\ \ startSend1((msgid, currentstate)\ :\ I\ \times$

$MSGSTATE) = let\ val\ new\_msgid = 1$
$val\ new\_currentstate = PaymentOrder$
$in\ (new\_msgid, new\_currentstate)\ end$

- a place $p \in P$ represents the protocol state, a transition $t \in T$ represents the message exchange consisting on an invocation of a Web service operation, and the initial marking M_0 represents the Web service initial state.

For instance a part of the Travel Agency Web service behavior is shown in Fig. 3.

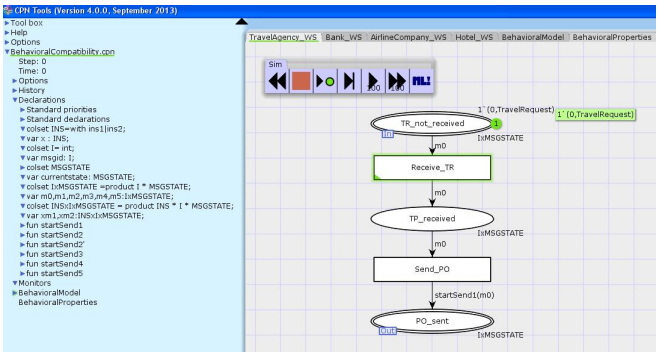

Fig. 3. A CPN modeling of a part of the Travel Agency Web service behavior

As we can see, each of the Travel Agency Web service operation is represented by a transition. The initial marking consists in the token in the place $TR\_not\_received$. The relations between operations are modeled by the firing rules of the CPN:

- $Receive\_TR$ is the first transition that can be fired if the token $(0, TravelRequest)$ is present in its input place.
- $Send\_PO$ will then be fired with the function $startSend1$ defined above.

We can now describe the CPN that models this part of the Travel Agency Web service behavior.

1) $\Sigma = \{INS, I, MSGSTATE, I \times MSGSTATE, INS \times I \times MSGSTATE\}$,
2) $P = \{TR\_not\_received, TP\_received, PO\_sent\}$
3) $T = \{Receive\_TR, Send\_PO\}$
4) $E(TR\_not\_received, Receive\_TR) = E(Receive\_TR, TP\_received) = E(TP\_received, Send\_PO) = m0, \ldots$
5) $M_0 = \{(0, TravelRequest)\ ++\ ins1\}$.

A finite execution of the protocol of the Travel agency Web service is defined by a sequence $M_0 \xrightarrow{Receive\_TR, msgid=0, currenstate=TravelRequest} M_1 \xrightarrow{Send\_PO, msgid=1, currenstate=PaymentOrder} M_2, \ldots$

2) Simulation: The above figure showed that the marking M_0 of the Travel Agency model changed to another marking M_1 after the occurrence of the transition *Receive_TR*. Another transition could be enabled and fire as a result of the new marking. This process of firing of a sequence of transitions is called *simulation*. Fig. 3. shows the simulation tool palette used for validating the Travel Agency Web service behavior. We note that simulations are also performed on CPNs composition step. In addition, simulations analyze a finite number of executions and help to validate the model by detecting and finding errors in the CPN model and demonstrates that the model works correctly. However, it is impossible to guarantee the correctness of the model with 100% certainly because all the possible executions are not covered [15]. This correctness will be analyzed in the second phase of our approach.

3) CPNs Composition: From the Web service behaviors that have been modeled on CPN models, we can now perform their composition using the concept of *sub-module* and the result will be a formal model that represents the Web service choreography called *behavioral model*. In this composition, the CPN models can be structured into a set of sub-modules to handle large specifications. These modules-pages interact with each other through a set of well-defined interfaces, in a similar way to programming languages. Fig. 4. shows the CPNs composition where we have four sub-modules (T, B, H, AC) representing respectively the four Web services behaviors (Travel Agency, Bank, Hotel, and Airline Company).

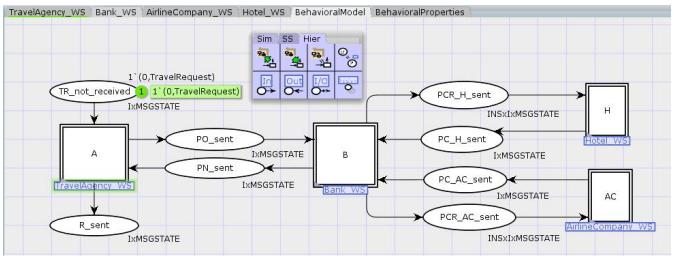

Fig. 4. CPNs Composition

For example, the two places *TR_not_received* and *PN_sent* represent *input ports* for the T sub-module. The two places *PO_sent* and *R_sent* are its *output ports*. This means that these places form the interface through which the T sub-module exchanges tokens with the other sub-pages. It will import tokens via the input ports and it will export tokens via the output ports. The composition of p sub-modules $N_1, \ldots N_p$ is denoted by $N_1 \oplus \cdots \oplus N_p$. Since we have composed our CPNs models representing the taken Web services, we can substitute each sub-module by its corresponding CPN. As said above, simulation is performed on the composition to validate it but, it is not sufficient to prove its behavioral compatibility. To do so, we perform the next phase.

B. Behavioral Properties Checking

From a generated behavioral model representing the Web service choreography, the behavioral properties checking can be performed through the verification of the message order properties. The notion of syntactical and semantic compatibility are preconditions of the following checking. Also, we consider the case where the component Web services in a choreography have correct behaviors. In this case, whether the composition can properly execute or not depends on the behavioral compatibility of its participating Web services.

Definition 5 (Behavioral Compatibility)**.** *Let* $N = N_1 \oplus N_p$ *be a CPN representing the behavioral model produced by*

 ISBN: 978-1-61208-337-7

the composition of p CPNs $N_1, \ldots, N_p$ representing the Web service models. Let $(i, j, request, m)$ denote transition labels for the sending of a request m from service i to service j and let $(i, j, answer, m)$ denote the transition label for the answer to this request from j to i. Then, N is behaviorally compatible with respect to message ordering if for all i, j, m, the following state formulas are satisfied by the initial configuration of N :

- $\mathsf{AG}(\langle(i, j, request, m)\rangle \rightarrow \mathsf{AF}(\langle(i, j, answer, m)\rangle))$, *meaning that any request is eventually followed by an answer, and*
- $\mathsf{AU}(\neg\langle(i, j, answer, m)\rangle, \langle(i, j, request, m)\rangle)$, *meaning that no answer is sent until a request has been sent first.*

Justification: We recall that each Web service is represented by a CPN, each Web service interaction (send or receive) is represented by a transition, and each exchanged message is represented by a color set of the token. Analyzing the behavioral compatibility of a Web service choreography is subject to verifying its correctness. This correctness is related to some qualitative requirements that are set on the order of the exchanged messages. We note that the second formula corresponds to the property given in Section II as example for the case study: *The payment confirmation will be sent by the Airline Company after it receives the payment confirmation request*. Both of the two formulas will be verified using a model checking technique based on SS. Thus, in this second phase, we have three related steps: *State Space Computation, ASK-CTL/SML Property Description, and Model Checking*.

1) State Space Computation: Our approach verifies the behavioral compatibility of a Web service choreography by using CPN Tools to automatically generate the transition system $\mathcal{T}_N$ associated with the choreography model. Only nodes reachable from the initial marking M_0 of the net and the associated transitions are kept by the tool. For our example above, the transition system $\mathcal{T}_N$ has 17 nodes (see Fig. 5.) representing the different markings, generated by all transitions:
$M_0 \xrightarrow{x=ins1,Receive\_TR,msgid=0,currentstate=TravelRequest} M_1$
$M_1 \xrightarrow{x=ins1,Send\_PO,msgid=1,currentstate=PaymentOrder} M_2$
and so on, up to
$M_{16} \xrightarrow{x=ins1,Send\_Response,msgid=5,currentstate=Response} M_{17}$

The transition system $\mathcal{T}_N$ can be used not only to obtain a standard report (including standard properties such as deadlock freeness) but also to verify ASK-CTL formulas like those defined for compatibility.

2) ASK-CTL/SML Properties Description: ASK-CTL formulas are used here to describe the behavioral properties to be checked. Let us deal with the behavioral property taken for our example (corresponding to a formula of the second type in Definition 5).

Behavioral Property: *The payment confirmation will be sent by the Airline Company after it receives the payment confirmation request.*

We rewrite the corresponding ASK-CTL formula into SML to obtain a concrete formalization of the property (see Table I). This formula is given by $\mathsf{AU}(\neg A_2, A_1)$ where A_1 denotes the characteristic predicate for the transition of receiving the payment confirmation request by the Airline Company Web service and A_2 denotes the characteristic predicate for the transition of sending the payment confirmation by the same Web service.

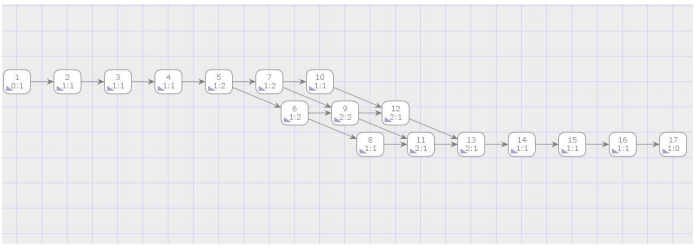

Fig. 5. Transition system of our behavioral model

TABLE I
SML FUNCTIONS FOR CHECKING THE BEHAVIORAL PROPERTY OF THE EXAMPLE

	SML Description
Functions and values declaration	$fun\ Arc1$ $a = (Bind.BehavioralModel'Receive\_PCR\_AC$ $(1, \{xm2 =$ $(ins1, 2, PaymentConfirmationRequestAC)\})$ $= ArcToBE\ a);$ $fun\ Arc2$ $a = (Bind.BehavioralModel'Send\_PC\_AC$ $(1, \{xm2 =$ $(ins1, 2, PaymentConfirmationRequestAC)\})$ $= ArcToBE\ a);$ $valA1 = AF("Receive", Arc1);$ $valA2 = AF("Send", Arc2);$
Formula	$val\ myASKCTLformula =$ $FORALL\_UNTIL(NOT(A2), A1);$
Verification	$eval\_arc\ myASKCTLformula\ InitNode;$

In this description, A_1 is interpreted by:
$fun\ Arc1$
$a = (Bind.BehavioralModel'Receive\_PCR\_AC$
$(1, \{xm2 = (ins1, 2, PaymentConfirmationRequestAC)\});$
referring the variable $xm2$ of transition $Receive\_PCR\_AC$.

And A_2 is interpreted by:
$fun\ Arc2$
$a = (Bind.BehavioralModel'Send\_PC\_AC$
$(1, \{xm2 = (ins1, 2, PaymentConfirmationRequestAC)\});$
referring to the variable $xm2$ of transition $Send\_PC\_AC$.

The global formula $(FORALL\_UNTIL(NOT(A_2), A_1)$ holds if the $Payment\ Confirmation$ message is not sent by the Airline Company until the $Payment\ Confirmation\ Request$ has been sent. Note that $InitNode$ means the initial marking of the transition system.

3) Model Checking: Here, we adopt the model checking toolkit provided by CPN Tools 4.0 to check whether the generated behavioral model N meets the two conditions introduced in the behavioral compatibility definition (definition 5).

First, ASK-CTL module should be loaded in CPN Tools 4.0. The commands are shown in high part of Fig. 6. Then,

 ISBN: 978-1-61208-337-7

the SML property description is written and then evaluated by “evaluate ML” option in the simulation tool palette. The checking result is shown in the green part of Fig. 6.

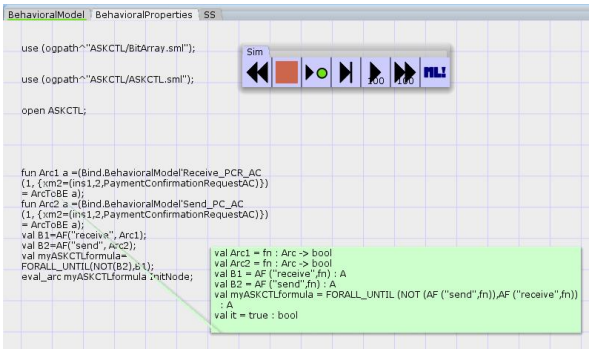

Fig. 6. Model Checking the behavioral property: true

We can see that the checking results returns true, which indicates that the behavioral model satisfies the property. This checking is not sufficient to say that the behavioral model is correct. We also need to check the two conditions given in the behavioral compatibility definition for all pairs (request, answer) in our modeled system. In our approach, if failures are detected then we must return to the first phase to correct these errors. For our example, we make some errors related to message order and in this case the checking results of our same taken property is given in Fig. 7. The correction is based on a behavior failure analysis that is done on exploring all property violation scenarios and pinpoints areas where modeling changes or revisions will be considered.

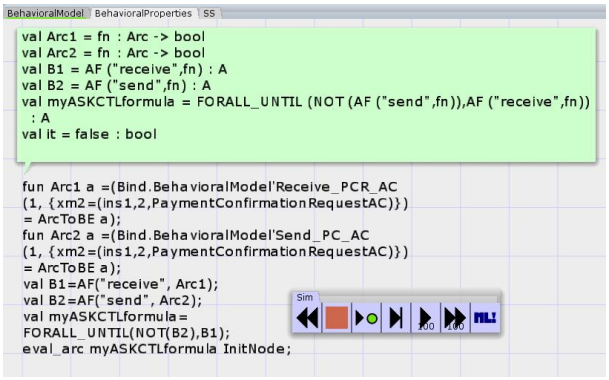

Fig. 7. Model Checking the behavioral property: false

Having shown that CPN based model checking of order property is feasible, we can then exploit the CPN Tools advanced graphical environment, to interactively simulate the actions performed in possible property violation scenarios. Behavior failure analysis is based on inspection of the terminal markings in all property violation paths. The simulation control functionality found in the CPN Tools 4.0 allows firing transitions with an interactively chosen transition. Thus, the actions included in the scenario of interest are easily reproduced and we can explore all possible behavior revision prospects to repair the detected property violation.

V. Related Work

To capture the behavior of service composition in some formal way, a variety of formal analysis techniques have been proposed. Most of them adopt a formal model such as PNs or Finite State Machines (FSM) or pi-calculus to express service behavior in a service orchestration and then utilize its theories and tools to accomplish the automatic verification. For example, Lucchi and Mazzara [16] propose an approach that analyzes service orchestration using WS-BPEL and the formalism pi-calculus. Benatallah et al. [17] propose an approach that analyzes the behavioral compatibility and the similarity of Web services. Hamadi et al. [18] propose an algebra of PNs to analyze the behavioral compatibility of Web services. The orchestration is modeled by the use of simple operators such as arbitrary sequence and more complex operators like iteration. Also, Tan et al. [19] propose an approach to analyze the compatibility of two services by translating their BPEL abstract processes into CPNS and check if their composition violates the constraints imposed by either side.

Compared to the works listed above, the approach proposed in [20] verifies service choreography by checking not only deadlock-freeness but also other properties, such as liveness and other specific properties. This approach is based on the automata formalism for modeling and on model checking for the analysis of behavioral compatibility and the satisfaction of temporal constraints: timing conflicts that may arise in a choreography can be detected. Another example that investigated choreography is [21], where Martnes et al. propose a PN based approach to model and analyze the behavioral compatibility of Web services, initially described by BPEL processes. Each selected BPEL process is transformed into a BPN. Then, the corresponding BPN models are composed, and the deadlock-freeness of the resulting net has to be proven.

In contrast to these works, our paper focuses not only on automatically reasoning about deadlock freeness, but also on message ordering properties. In addition, our verification is done at design time while current approaches are specific to a given programming language and only focus on the verification of already implemented composite services. The benefit of our approach is that the composition specification is proven to be correct before its implementation with a programming language such as BPEL. Few works has been done, to the best of our knowledge, in this research direction. For example, Achilleos et al. [22] propose an approach that combines Model Driven Architecture (MDA) and PNs to provide design, verification and code generation. Recently, in [23], a MDA for creating consistent service orchestrations is presented. Service execution and interaction are described with a high-level model in terms of extended PNs notation. Also, recently, Dumez et al. [24] propose a MDA approach to specify, verify and implement service composition using existing specification and implementation languages. To support the formal verification of the composition, a translation of the composition workflow model is done into a Language

 ISBN: 978-1-61208-337-7

of Temporal Ordering Specification (LOTOS [25]) formal specification. The CADP [26] tool-set is then used to verify the composition via its LOTOS specification. Our work has a similar objective, adopting instead CPNs to formalize the behaviors and interactions of services. This model is well suited to specify service composition due to its compositionality properties. Moreover, it uses CPN Tools, providing the designer with the ASK-CTL toolkit that is expressive enough to describe message ordering.

Our paper presents a formal approach that goes beyond checking for deadlock-freeness as proposed by the majority of related work. We note that our approach has a disadvantage since it is based on state space analysis that presents the *state explosion problem*. To address this problem, we can use reduction techniques that are supported by CPN Tools 4.0.

VI. Conclusion and Future Work

CPNs enhance classical PNs with commonly agreed upon extensions such as data and hierarchy. The resulting modeling language is highly expressive and is supported by CPN Tools 4.0, a recent powerful software tool for the modeling and analysis of CPNs. This paper used an example to explain the behavioral compatibility that is analyzed using CPNs during the early design phase of choreography, thus avoiding iterative cycles between the choreography implementation and the compatibility analysis. The interest of our proposed approach lies in the clear presentation of the analysis model and readiness for its implementation. We have demonstrated how to use CPNs to model and compose the Web service behaviors and how to use CPN Tools 4.0 to analyze their behavioral compatibility basing on ASK-CTL and model checking.

In future work, we will extend our model to allow the performance analysis in terms of quantitative timing constrains. Our new timed model will be based on Timed CPN [7], also supported by CPN Tools 4.0, and its analysis will be done by simulations that allow performance analysis. In addition, we plan to extend our approach of verification by using MDA in order to have a development process of service choreography that is based of CPNs and Timed CPNs.

References

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: A Research Roadmap," *International Journal of Cooperative Information Systems*, vol. 17, no. 02, pp. 223–255, 2008, ISSN: 0-2-1-8-8-4-3-0.

[2] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. Ferguson, *Web Services Platform Architecture: SOAP, WSDL, WS-policy, WS-addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR, 2005, ISBN: 013-14-88-74-0.

[3] Y. Zhu and H. Gao, "A Novel Approach to Generate the Property for Web Service Verification from Threat-Driven Model." *Applied Mathematics & Information Sciences*, vol. 8, no. 2, 2014.

[4] G. Piccinelli, W. Emmerich, C. Zirpins, and K. Schutt, "Web service interfaces for inter-organisational business processes an infrastructure for automated reconciliation," in *Enterprise Distributed Object Computing Conference*. IEEE, 2002, pp. 285–292.

[5] Y. Hongli, Z. Xiangpeng, Q. Zongyan, P. Geguang, and W. Shuling, "A formal model for web service choreography description language (ws-cdl)," in *IEEE International Conference on Web Services, Chicago, IL, USA*. Plattner Institute, University of Potsdam, German, and Queensland University, 2006, pp. 893–4.

[6] M. De Backer, M. Snoeck, G. Monsieur, W. Lemahieu, and G. Dedene, "A Scenario-Based Verification Technique to Assess the Compatibility of Collaborative Business Processes," *Data & knowledge engineering*, vol. 68, no. 6, pp. 531–551, 2009.

[7] K. Jensen and L. M. Kristensen, *Coloured Petri Nets: Modelling and Validation of Concurrent Systems*. Springer Berlin Heidelberg, 2009.

[8] A. Cheng, S. Christensen, and K. Mortensen, "Model Checking Coloured Petri Nets-Exploiting Strongly Connected Components," *DAIMI Report Series*, vol. 26, no. 519, 1997.

[9] M. Wastergaad, "CPN tools 4: Multi-Formalism and Extensibility," in *Application and Theory of Petri Nets and Concurrency*. Springer, 2013, pp. 400–409, Jose-Manuel, C. and Jorg, D., Ed., ISBN: 978-36-42-38-69-61, ISSN: 0-3-0-2-9-7-4-3.

[10] Z. Maamar, B. D., G. Mostefaoui, S. Subramanian, and Q. Mahmoud, "Toward Behavioral Web Services Using Policies," *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, vol. 38, no. 6, pp. 1312–1324, 2008, ISSN: 1-0-8-3-4-4-2-7.

[11] W. M. P. Van der Aalast, C. Stahl, and M. Wastergaard, "Strategies for Modeling Complex Processes Using Colored Petri Nets," in *Transactions on Petri Nets and Other Models of Concurrency VII*. Springer, 2013, pp. 6–55, Jensen, K. and Van der Aalast, W. M. P. and Balbo, G. and Koutny, M. and Wolf, K., Ed., ISBN: 978-36-42-38-14-23, ISSN: 0-3-0-2-9-7-4-3.

[12] S. Christensen and K. Mortensen, "Design/CPN ASK-CTL manual," *University of Aarhus. 0.9 edn*, 1996.

[13] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 8, no. 2, pp. 244–263, 1986.

[14] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual*. Pearson Higher Education, 2004.

[15] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems," *International Journal on Software Tools for Technology Transfer*, vol. 9, no. 3-4, pp. 213–254, 2007, ISSN: 1-4-3-3-2-7-7-9.

[16] R. Lucchi and M. Mazzara, "A pi-calculus based semantics for ws-bpel," *The Journal of logic and Algebraic Programming*, vol. 70, no. 1, pp. 96–118, 2007.

[17] B. Benatallah, C. F., and F. Toumani, "Analysis and Management of Web Service Protocols," in *Conceptual Modeling–ER 2004*, 2004, pp. 524–541, Atzeni, P. and Chu, W. and Lu, H. and Zhou, S. and Ling, T.W., Ed., ISBN: 978-35-40-23-72-35, ISSN: 0-3-0-2-9-7-4-3.

[18] R. Hamadi and B. Benatallah, "A Petri net-based model for web service composition," in *Proceedings of the 14th Australasian database conference-Volume 17*. Australian Computer Society, Inc., 2003, pp. 191–200, ISBN: 090-99-25-95-X.

[19] W. Tan, Y. Fan, and M. Zhou, "A petri Net-Based Method for Compatibility Analysis and Composition of Web Services in Business Process Execution Language," *Automation Science and Engineering, IEEE Transactions on*, vol. 6, no. 1, pp. 94–106, 2009, ISSN: 1-5-4-5-5-9-5-5.

[20] N. Guermouche and C. Godart, "Characterizing Compatibility of Timed Choreography," *International Journal of Web Services Research*, vol. 8, no. 2, pp. 1–28, 2011.

[21] A. Martnes, S. Moser, A. Gerhardt, and K. Funk., "Analyzing Compatibility of Bpel Processes," in *Proceedings of the International Conference on Internet and Web Applications and Services/Advanced International Conference, 2006*. IEEE, 2006, pp. 147–147, ISBN: 076-95-25-22-9.

[22] A. Achilleos, K. Yang, N. Georgalas, and M. Azmoodech, "Pervasive Service Creation Using a Model Driven Petri Net Based Approach," in *Wireless Communications and Mobile Computing Conference, 2008. IWCMC'08. International*. IEEE, 2008, pp. 309–314.

[23] G. Grossmann, M. Schrefl, and M. Stumptner, "Design for service compatibility," *Software & Systems Modeling*, vol. 12, no. 3, pp. 489–515, 2013, ISSN: 1-6-1-9-1-3-6-6.

[24] C. Dumez, M. Bakhouya, J. Gaber, M. Wack, and P. Lorenz, "Model-Driven Approach Supporting Formal Verification for Web Service Composition Protocols," *Journal of Network and Computer Applications*, vol. 36, no. 4, pp. 1102–1115, 2013.

[25] H. Garavel and J. Sifakis, "Compilation and verification of lotos specifications," in *PSTV*, vol. 10, 1990, pp. 359–376.

[26] J. C. Fernandez, H. Garavel, A. Kerbrat, L. Mounier, R. Mateescu, and M. Sighireanu, "CADP a Protocol Validation and Verification Toolbox," in *Computer Aided Verification*. Springer, 1996, pp. 437–440.

 ISBN: 978-1-61208-337-7