Model Checking Verification of Web Services Composition

Abdallah Missaoui[1], Zohra Sbaï[1] and Kamel Barkaoui[2]

[1] Ecole Nationale d'Ingénieurs de Tunis, BP. 37 Le Belvédère, 1002 Tunis, Tunisia

[2] Conservatoire National des Arts et Métiers, Rue Saint Martin, 75141 Paris, France

Abstract. Web services composition is becoming very important in today's service oriented business environment. Different services frequently have semantic inconsistencies which may lead to the failure of the services composition. In order to verify the correctness of the Web Services composition, we present a method for analyzing and verifying interactions among web services. We model web service composition based on special class of Petri nets: open workflow nets. We translate this composition to Promela, a source language of SPIN model checker, designed to describe communicating distributed services. At the requirements level, model checking is used to validate the specification against a set of formulae specified into LTL which are used to verify constraints satisfaction of web services composition.

1 Introduction

Founded on standard protocols, Web service is a kind of software interface and platform, which is described, published and invoked through the Internet. Web services have the capability of implementing distributed applications. Recently, integrating various services distributed becomes a valuable issue in current research on Web services.

The behaviour of a service composite may become very complex due to the complex of the communication between partners. The tasks that are performed within a service generally depend strongly on the interaction that has taken place. Thus, it is very important to decide whether all services interact properly. At present, many industry and academia researchers are paying attention to find some methods to detect these behaviour problems such deadlock.

Model checking is a verification technique that explores all possible system states in a brute-force manner. A model checker, the software tool that performs the model checking, examines all possible system scenarios in a systematic manner. In this way, it can be shown that a given system model truly satisfies a certain property. There are many powerful model checkers such as NuSMV [4], BLAST [5] and SPIN [6, 7].

This paper proposes to use the software model checking techniques for the verification of web services composition. We adapt Petri nets to describe services interaction and use SPIN model checker as a verification engine. First, a composed system model is written in Promela (PROcess MEta LAnguage) that describes the behaviour of the web services composition. Then, correctness properties that express requirements on

Missaoui A., Sbaï Z. and Barkaoui K. (2010).
Model Checking Verification of Web Services Composition.
In *Proceedings of the 4th International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing*, pages 114-126
DOI: 10.5220/0003051501140126

the system's behaviour are specified in Linear Temporal Logic (LTL). By efficient exploration of the complete set of states generated from the Promela specification, SPIN verify LTL formulae corresponding to properties of web service composition.

The rest of this paper is organized as follows. Section 2 presents preliminaries. Section 3 focuses on Petri net model of web service composition as well as our Promela implementation of this composition. In section 4, We formulate in LTL some soundness properties to be verified by SPIN. In section 5, we study an example of web service composition. We discuss in section 6 some related work. Section 7 concludes the paper.

2 Preliminaries

2.1 Workflow Process Modelling

Petri nets are a well founded process modeling technique that have formal semantics. They are one of the best known techniques for specifying business processes in a formal and abstract way. The semantics of process instances ensuing from process models described in Petri nets are well defined and not ambiguous.

We choose to adopt open workflow nets [17] for web service modelling. Open workflow nets result from the application of Petri nets to workflow management. In fact, Petri nets have been used for their formal semantics, graphical nature, expressiveness, analysis techniques and tools.

In the rest of this section, we present first the basic definitions and notations of Petri nets [2] used in this work. Then we highlight the notion of open workflow nets.

Petri Nets

A Petri net is a 4-tuple $N = (P, T, F, W)$ where *P* and *T* are two finite non-empty sets of places and transitions respectively, $P \cap T = \emptyset$, $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation, and $W : (P \times T) \cup (T \times P) \rightarrow \mathbb{N}$ is the weight function of *N* satisfying $W(x, y) = 0 \Leftrightarrow (x, y) \notin F$.

If $W(u) = 1 \ \forall u \in F$ then *N* is said to be ordinary net and it is denoted by $N = (P, T, F)$.

For all $x \in P \cup T$, the preset of *x* is ${}^\bullet x = \{y | (y, x) \in F\}$ and the postset of *x* is $x^\bullet = \{y | (x, y) \in F\}$.

A marking of a Petri net N is a function $M : P \rightarrow \mathbb{N}$. The initial marking of N is denoted by M_0.

A transition $t \in T$ is enabled in a marking M (denoted by $M[t\rangle$) if and only if $\forall p \in {}^\bullet t : M(p) \geq W(p, t)$. If transition t is enabled in marking M, it can be fired, leading to a new marking M' such that: $\forall p \in P : M'(p) = M(p) - W(p, t) + W(t, p)$.

The firing is denoted by $M[t\rangle M'$. The set of all markings reachable from a marking M is denoted by $[M\rangle$.

For a place *p* of *P*, we denote by M_p the marking given by $M_p(p) = 1$ and $M_p(p') = 0$ $\forall p' \neq p$.

Petri nets are represented as follows: places are represented by circles, transitions by boxes, the flow relation is represented by drawing an arc between *x* and *y* whenever (x, y) is in the relation, and the weight function labels the arcs whenever their weights

are greater than 1. A marking *M* of a Petri net is represented by drawing $M(p)$ black tokens into the circle representing the place *p*.

Open Workflow Nets (oWF-nets)
In this paper, we choose to adopt a special class of Petri-nets which is open workflow-nets (oWF-net), for web service modeling. It generalized the classical workflow nets [1] by introducing an interface for asynchronous messages with partners. A petri net $N = (P, T, F)$ is called an oWF-net, if:

i. P is composed from disjoint sets: internal places P_N, input places P_I and output places P_O;
ii. For all transition $t \in T : \ p \in P_I \, (resp. \, p \in P_O) \, implies \, (t, p) \notin F \, (resp. \, (p, t) \notin F)$;
iii. F does not contain cycles.

In all examples in this paper, an oWF-net are initially one token in start place (no tokens in other places including the interface places).

Open workflow nets allow diverse analysis methods of business process. Moreover, the explicit modeling of interface allows the verification and behaviour analysis of web service composition.

2.2 Model Checking Techniques

Model checking is a verification technique that explores all possible system states in a brute-force manner. Similar to a computer chess program that checks possible moves, a model checker, the software tool that performs the model checking, examines all possible system scenarios in a systematic manner. In this way, it can be shown that a given system model truly satisfies a certain property. It is a real challenge to examine the largest possible state spaces that can be treated with current means, i.e., processors and memories.

There are many powerful model checkers such as NuSMV, BLAST and SPIN.

The SPIN model checker is a system that can verify models of computerized systems. The name SPIN was originally chosen as an acronym for Simple Promela INterpreter. It can be used in two basic modes: as a simulator and as a verifier. In simulation mode, SPIN can be used to get a quick impression of the types of behavior that are captured by a system model, as it is being built. Some optimization techniques, e.g., partial order reduction and graph encoding, are available to help reduce the usage of CPU time or memory space.

2.3 Linear Temporal Logic

SPIN checks properties formulated with Liner Temporal Logic. An LTL formula f may contain any lowercase propositional symbol p, combined with unary or binary, Boolean and/or temporal operators. Propositional and temporal operators are presented in figure 1.

The semantics of a formula is given in terms of computations and the states of a computation. The atomic propositions of temporal logic can be evaluated in a single state independently of a computation.

Propositional Operators		
&& conjunction		$\|\|$ disjunction
$->$ implication		! Negation
Temporal Operators		
[] always	$\diamond$ eventually	$\bigsqcup$ until

Fig. 1. Propositional and Temporal Operators.

3 Web Services Composition

In this section, we begin with a specification of services in terms of open Workflow net and Promela language. Then, we model the composition of web services and we give a verification method by means of model checking techniques.

The formalism of open Workflow net is meant to model complex, distributed computing systems with well defined semantics. Web services composition can be checked against requirements such as the mentioned above. The properties that are to be checked need to be formulated based on temporal logic, relate to the Promela specification of workflow representation of web service composition. SPIN Model checker will evaluate these LTL formulae.

3.1 Modeling Web Service Composition

Using oWF-nets, we can model business process. We can furthermore compose two or many oWF-nets and describe the web services interaction with partners.

The communication between two oWF-nets is based on interface places (unidirectional). Given two oWF-nets N1 and N2, their composition is the result of merging one or many input places of one oWF-net with suitable output net of the other place and vice versa. Each oWF-net shares only interface places with other oWF-nets.

Fig 2.a shows the composition of two oWF-nets: Producer and Consumer. Each of which has one input place and one output place. Shared places which model the interfaces are represented by dashed lines. The consumer of the left side sends an order message to the producer in order to execute its task. In the right side, the producer executes the task and then sends a response to consumer.

3.2 Promela Implementation

Like structured programming languages, in Promela we can use variables and subroutines such as $proctype$. The proctype construct is used for the declaration. It can be used to declare process behavior is to be executed completely deterministically. Each process behavior must be declared before process instantiation. This instantiation can be done either with the run operator, or with the prefix $active$ that can be used at the time of declaration. The types of variables are sets of integer in order to guarantee that the program has finite state space. This section presents a method to use the SPIN model checker for the representation of the web services composition and analysis. The basic idea is just to translate a composition of oWF-nets into Promela source program. web services models communicate through interface places modeled by variables. We

adopt an approach of oWF-net introduced by [17] to model web services composition. Workflow module defines its internal behaviour without interface places that are used to represent message flow between partners.

Based on the definition of workflow module, we present a web service specification in Promela language. This specification is described in terms of the marking of places and the firing count of transitions. Hence, we use the following conventions:

- Places are represented by an array PL of integers with length equal to the number of places. This array contains initially zero in each element.
- Transitions are modelled by an array TR of integers (initialized to zero) with length equal to the number of transitions.

The behaviour of a web service can be described in terms of system states and their changes. A marking is initialized to M_i and it is changed according to the following transition rule: A transition t is enabled if each input place p of t is marked. Moreover, an enabled transition t may or may not fire, and a firing of t removes one token from all $p \in {}^\bullet t$ and adds one token to each output place of t. These concepts are translated in the corresponding Promela description of a workflow module as follows: The firing of a transition consists on decreasing by 1 the integer corresponding to each place $p \in {}^\bullet t$ in the array PL and increasing by 1 the elements of PL corresponding to the output places ($p \in t^\bullet$). Although, we increase by 1 the element of TR corresponding to the transition t to mark that t is fired once.

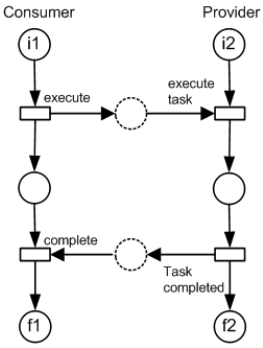
These modifications are made by the macro $add1$, $add2$, ..,$addS$, $remove1$, $remove2$, .., and $removeK$ where S is the maximum number of possible input places and K is the maximum number of possible output places.

To fire a transition t which has I input places and J output places, we call these macros with the appropriate arguments in order to achieve the following actions:

1. $removeI(p_1, p_2, .., p_I)$ destructs one token from each input place. It tests if these input places are marked ($PL[index\ of\ p_j] > 0, 1 \leq j \leq I$) and if this condition is satisfied, it removes one token from each of its parameters.
2. We increase the element corresponding to t in TR.
3. And finally $addJ(p_1, p_2, .., p_J)$ creates one token in each output place.

Complex inter-organizational business processes are structured as a set of communicating elementary services. A service represents a self contained software unit that offers an encapsulated functionality over a well defined interface.
Several languages, such as BPEL [3], are used to describe the composition of services. The behaviour of a service composite may become very complex due to the nature of the communication between partners. The tasks that are performed within a service generally depend strongly on the interaction that has taken place. Thus, analyzing the behaviour of a service on the one hand and the verification of its interaction with partners on the other hand are very important to perform and improve inter-organizational business processes.
Web services composition is modeled based on oWF-nets. Interface places are represented, in our specification, by an array I of integers with length equal to the number of places in web services composite. Each element of this array, that initialized to zero,

perform the communication between services partners, and for each place, when one service increment element of I (add token in interface place), another partner consume a token and decrement this element by 1. Figure 2.a shows an example of web service



(a)

```
#define PLACEC 3
#define TRANSITIONC 2
#define PLACEP 3
#define TRANSITIONP 2
#define INTERFACE 2
#define remove1(x1)(x1>0)->x1--
#define remove2(x1,x2)
(x1>0 && x2>0)->x1--; x2--
#define add1(x1) x1++
#define add2(x1,x2) x1++; x2++
int M2[PLACEC]; int X2[TRANSITIONC];
int M1[PLACEP]; int X1[TRANSITIONP];
int I[INTERFACE]
proctype Consumer()
{M2[0]=1;
do
:: atomic{remove1(M2[0])->X2[0]++;
          add2(M2[1],I[0]) }
:: atomic{remove2(M2[1],I[1])-> X2[1]++;
          add1(M2[2]) }
od
}
```

```
proctype Producer()
{M1[0]=1;
do
:: atomic{remove2(M1[0],I[0])->X1[0]++;
          add1(M1[1]) }

:: atomic{remove1(M1[1])->X1[1]++;
          add2(M1[2],I[1]) }
od
}

init
{
atomic{
        run Consumer();
        run Producer();
      }
}
```

(b)

Fig. 2. The process with two services.

composition based on two oWF-nets. The consumer communicates with producer by means of two interface places. The figure 2.b represents the Promela description of this composed system.

4 Verification

SPIN allows Promela systems to be simulated, either step-by-step or randomly. Additionally, it is possible to execute an exhaustive simulation by generating the complete state-space of the system. This allows us to verify if the system requirements are satisfied.

In web service architecture, by composing various services, complex activities (e. g., inter-organizational business processes) can be realized. Hence, the correct interplay of distributed services is crucial to accomplish a common goal. The flow should have safety properties such as deadlock freedom. At the same time, the flow should satisfy some properties since it is executed in an environment.

In this section, we discuss how to verify a given composed oWF-net specification using the SPIN model checher. The input language of SPIN is Promela, which is a language for modeling web services composition. SPIN verify LTL properties of Promela specification. We implement the verification process in three steps : (1)we model the composed web services based on oWF-net; (2)we translate this composed system to Promela processes with communication based on an interface array; (3)we define properties in LTL and we check if the system satisfies or not these properties.

The requierement needs to be formulated as LTL formulas. All logical variables used in the formula must be defined by $define$ macro. Using logical variables integrated in TLT formulae, all properties are verified on the fly. It is not necessary to generate the whole state space to detect errors. However, to verify that properties are satisfied, the whole state space may be generated.

4.1 Termination

Several Technologies [9–11] have been proposed in order to automatically compose Web services to perform some desired task. Regardless of how the compositions originated, we are interested here in describing and verifying properties of these services by simulating their executions under different input conditions. Deadlock is a condition which states that the composed system is deadlocked, i.e. it is neither terminated nor more activities could be executed.

The language source used by SPIN (Promela) describe the behaviour of a web services composition based on interactions between the service and its partners. SPIN can simulate and validate a model in Promela language and it can accept constraints described in LTL formulae.

```
Never_claim :
never {    /* !(<>p) */
accept_init:
T0_init:
        if
        :: (! ((p))) -> goto T0_init
        fi;
}
```

Fig. 3. Never-claim corresponding to F.

The formula $F : \Diamond p$ (p is declared in Promela source as $(M1[index\ of\ f1] \geq 1$ && $M2[index\ of\ f2] \geq 1)$) allows us to check if the process, given in figure 2, completes successfully. First, SPIN translates a negative form of formula into a never-claim statement. Second, SPIN validates the model S with this statement. The never-claim (generated by SPIN) corresponding to formula F is given in figure 3.

Never claim is the Promela model of the Büchi automaton(figure 4) corresponding to the LTL formula F. It is used to represent a property that should never be satisfied during the execution of a model. Then, if S contains an acceptance cycle then a counter-example to F exists and this proves that the model does not satisfy the property.

4.2 Soundness

Composing web services and guarantying the correctness of the design is an important and challenging problem in web services.

An oWF-net is a more suitable model for analysis and verification of web services composition. Several Correctness criteria, like liveness and termination, should be valid for single process and for the combined model. In this section, we give a method to verify some properties of web services composition. For each of them, we define the property of the composed oWF-nets. We also give an LTL formula that allows to verify it based on corresponding Promela specification of web services composition.

A workflow process is sound if, for any case, termination is guaranteed, there are no dangling references, and deadlocks and livelocks are absent. This dynamic property is decidable and it can be checked in polynomial time. Soundness property is proved in [18] equivalent to liveness and boundedness, thus it can be verified by standard Petri net methods.

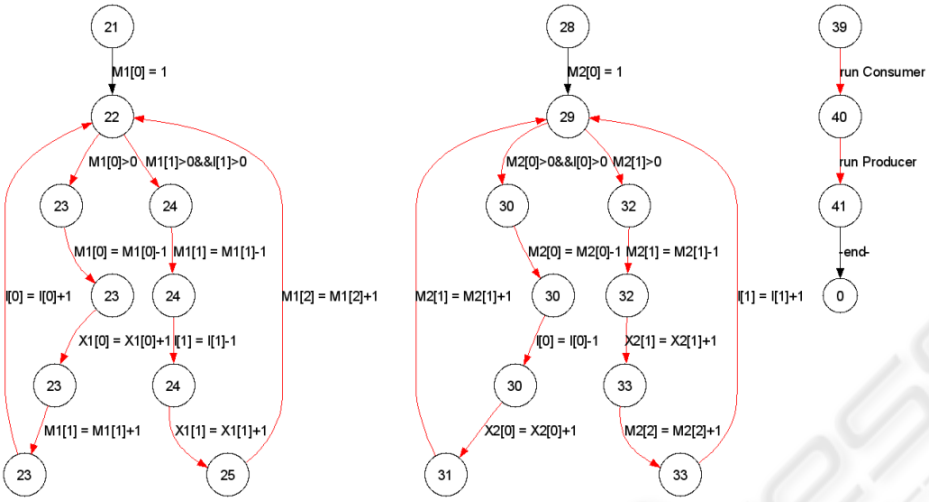


Fig. 4. A büchi automata.

The extension of this soundness definition in the context of web service composition via oWF-nets is given in the following definition 1. The overall net describing the composition can be seen as a classical petri net with a specific initial marking and its soundness is given in terms of states evolution and transitions firing. Thus the verification of this property is decidable.

Definition 1. *(Soundness)*
Let $N_1 = (P_1, T_1, F_1)$, $N_2 = (P_2, T_2, F_2)$,..., *and* $N_x = (P_x, T_x, F_x)$ *be* x *oWF-net. The composed oWF-nets* $WC = (P, T, F)$ *from* $N1$, $N2$ *and* Nx, *is called sound if and only if the following three requirements are satisfied:*

- *i* Termination: *For each reachable marking (reachable from* $M_0 = [i_1, i_2, .., i_x],$*) the final marking* $M_f = [f_1, f_2, .., f_x]$ *is reachable;*
- *ii* Proper completion: *For each reachable marking, if* $M \geq M_f$ *holds then* $M = M_f$*;*
- *iii* No dead transition: *It makes certain that starting from the initial state (just one token in every initial place), it is always possible to reach the state with one token in every final place.*

We can define LTL formulae based soundness of web service composition by studying the three sub-properties: Completion, Proper Completion and Deadlock-freedom:

i $\Diamond(\overset{i=x}{\underset{i=1}{\&\&}} M[index\ of\ f_i] \geq 1)$

ii $\Box(\overset{i=x}{\underset{i=1}{\&\&}} M[index\ of\ f_i] \geq 1) \Rightarrow (M = [f_1, f_2, .., f_x])$

iii $\neg(\Box(\underset{t \in T}{\|} X[index\ of\ t] == 0))$

To verify soundness of web service composition, we have to check the three preceding formulae.

Weak soundness properties are introduced by Martens [8] in the context of web service composition. We can allow service that exhibits a proper behaviour but not all tasks. The overall Composed system is weak sound, because the final state can be reached from each state reachable from the initial state, and when the final state M_f is reached, no other token remains in the net.

Using Petri nets notation, we give the following definition of weak soundness.

Definition 2. *(weak soundness)*
a composed system $N = (P, T, F)$ is called weak sound if and only if:

i. For each reachable marking (reachable from M_0) the final marking f is reachable.
ii. For each reachable marking, if $M \geq M_f$ holds then $M = M_f$.

Based on LTL formulae, corresponding to Termination and Proper Completion, defined in this section, we can verify weak soundness properties of the web service composition.

5 Case Study

Figure 5 presents an example of web service composition. The overall flow of this example is as follows: A traveler will plan her trip. When the traveler is finished with specifying the details of the trip plan, she sends this information together with the prepared details (example of details concern the list of participants) to the travel agent. Next, the traveler will await the submission of the electronic tickets as well as the final itinerary for the trip. When the agent receives the traveler's trip order, he will determine the legs for each of the stages, which includes an initial seat reservation for each of the participants as well as the rate chosen. To actually make the corresponding ticket orders the agent submits these legs together with the information about the credit card to be charged to the airline company.

Then, the agent will wait for the confirmation of the flights, which especially includes the actual seats reserved for each of the participants. This information is completed into an itinerary, which is then sent to the traveler.

When the airline receives the ticket order submitted by the agent, the requested seats will be checked. After that, the credit card will be charged, and the updated leg information is sent back to the agent as confirmation of the flights. After that, the airline sends the electronic tickets (by e-mail) to the traveler. Information about the recipient of the tickets is passed to the agent as well as to the airline.

In this example, we propose to draw its Promela model as follows: we begin with specifying each process as a single process defined by $proctype$ construct. The processes describing the work of the traveler, the agent and the airline are given respectively in the following source code.

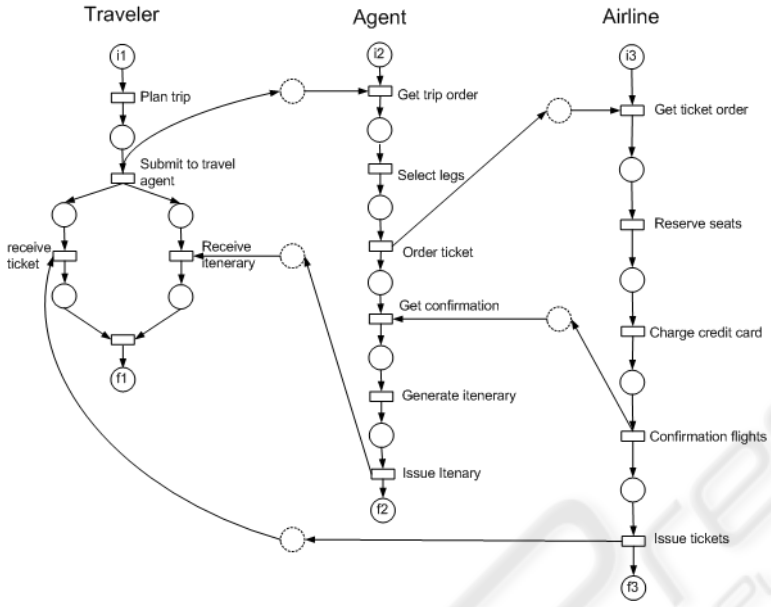


Fig. 5. Travel Plan Example.

```
proctype Traveler()
{PL1[0]=1;
do
::atomic{inp1(PL1[0]) -> TR1[0]++;out1(PL1[1])                }
::atomic{inp1(PL1[1]) -> TR1[1]++;out3(PL1[2],PL1[3],I[0])}
::atomic{inp2(PL1[2],I[4]) -> TR1[2]++;out1(PL1[4])       }
::atomic{inp2(PL1[3],I[1]) -> TR1[3]++;out1(PL1[5])       }
::atomic{inp2(PL1[4],PL1[5]) -> TR1[4]++;out1(PL1[6])     }
od }
proctype Agent()
{PL2[0]=1;
do
::atomic{inp2(PL2[0],I[0]) -> TR2[0]++;out1(PL2[1])     }
::atomic{inp1(PL2[1]) -> TR2[1]++;out1(PL2[2])     }
::atomic{inp1(PL2[2]) -> TR2[2]++;out2(PL2[3],I[2])     }
::atomic{inp2(PL2[3],I[3]) -> TR2[3]++;out1(PL2[4])     }
::atomic{inp1(PL2[4]) -> TR2[4]++;out1(PL2[5])     }
::atomic{inp1(PL2[5]) -> TR2[5]++;out2(PL2[6],I[1])     }
od }
proctype Airline()
{PL3[0]=1;
do
::atomic{inp2(PL3[0],I[2]) -> TR3[0]++;out1(PL3[1])     }
::atomic{inp1(PL3[1]) -> TR3[1]++;out1(PL3[2])     }
::atomic{inp1(PL3[2]) -> TR3[2]++;out1(PL3[3])     }
::atomic{inp1(PL3[3]) -> TR3[3]++;out2(PL3[4],I[3])     }
::atomic{inp1(PL3[4]) -> TR3[4]++;out2(PL3[5],I[4])     }
od }
```

Note that in each process, the firing of a transition leads to add tokens not only in its own places but also in interface ones. The interface places are modelled by an array called I of ni elements with ni is the number of these places. The initial process, declared by $init$, initializes global variables, and instantiates processes via the run statement. This statement is used as a unary operator that takes the name of a process type. An executing process disappears again when it terminates (i.e., reaches the end of the body of its process type declaration), but not before all processes that it started have terminated. The sequence of composed processes has to be executed as one indivisible unit, non-interleaved with any other processes. Hence, we prefix the sequence of processes instantiations by $atomic$.

```
init
{
atomic{
    run Traveler();
    run Agent();
    run Airline()
    }
}
```

To verify soundness of web services composition, three requirements need to be formulated as LTL formula. We introduce logical variables that take part from the LTL formula in the source code. Using the Promela source and the logical variable definitions, SPIN evaluates the LTL formula. The web service composition presented in fig 5 is sound because it satisfies the three formulae (termination, proper completion and deadlock-freedom).

For example, the evaluation of a formula F ($<> p$) of termination (p: $PL1[6] > 0 \,\&\&\, PL2[6] > 0 \,\&\&\, PL3[5] > 0$) have resulted in positive answers given that the number of errors returned by SPIN is 0. This positive result guarantees that the Promela model meets this requirement. All states have been explored by SPIN and the result generated shows that the number of states explored is 24.

6 Related Works

Many works such as [8, 16, 15] introduce formal method based on Petri nets for describing and verifying web service composition. In [15], semantics of Petri nets is defined by mapping BPEL process to a Petri net. A formal model of BPEL can be generated and allows the verification techniques developed for Petri nets to be exploited in the context of BPEL processes. Lohmann [16] focuses on the problem of analyzing the interaction between WS-BPEL processes. A BPEL process is seen as an open workflow net by an interface specifying the interactional behaviour of this process with its partners. The framework developed in this context allows the generation of compact Petri net models and gives a formal analysis of processes behaviour and a verification of controllability.

Some research efforts have already been proposed to use model-checking techniques for web service verification. Nakajima [12] describes a method based on model checking to verify web service flow description. The language adopted to describe web service composition is Web Services Flow Language (WSFL). The model checker used in their verification engine is SPIN. This paper presents means to translate WSFL primitives into PROMELA. In [13], Nakajima proposed to extract behaviour specification from BPEL processes to a variant of Extended Finite-state Automaton. Then, Automaton model is translated to PROMELA and is analyzed by SPIN model checker. Nevertheless, not all BPEL processes can be analyzed i.e. it does not deal with the semantics of handlers such as exception or compensation.

Bao [14] provides a model checking method to Verify BPEL4People processes that can detect deadlocks and validate constraints based on LTL. A tool is developed to translate BPEL4People process to promela automatically but it didn't support all feature activities.

7 Conclusions

In this paper, we have presented a specification and verification method of web services composition based on model checking. The interactions among the partners participating to a web services composition are modeled by Open WorkFlow nets (OWF-nets), where the communication is ensured by interface places. Hence, we begun with converting OWF-nets to Promela: the model specification language used to define the relevant aspects of the system needed to verify it.

The formal model presented in this paper captures and checks the control flow and the dynamic behaviour based on Promela specification that describes the behaviour of the web services composition. After this step, we have specified in Linear Temporal Logic (LTL) correctness properties that express requirements on the system's behaviour. Especially, we have defined the soundness properties of the web service composition in terms of LTL formulae.

Once the formal model and the correctness properties are defined, SPIN model checker verifies LTL formulae corresponding to properties of web service composition, by efficient exploration of the set of states generated from the Promela specification.

We propose to extend this work by defining additional verification criteria which are suitable for the composition of web services such as controllability.

References

1. van der Aalst, W.M.P.: The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, vol. 8, (1998) 21-66
2. Barkaoui, K., Ben Ayed, R., Sbaï, Z.: Workflow Soundness Verification based on Structure Theory of Petri Nets. International Journal of Computing and Information Sciences (IJCIS), Vol. 5(1), (2007) 51–61
3. Lohmann, N., Massuthe, P., Stahl, C., Weinberg, D.: Analyzing interacting WSBPEL processes using flexible model generation. DKE 64(1), (2008) 3854
4. Cimatti, A., Clarke, E. M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R. and Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In Proceeding of International Conference on Computer-Aided Verification, (2002)
5. Henzinger, T. A., Jhala, R., Majumdar, R. and Sutre, G.: Software Verification with Blast. In Proceedings of the 10th SPIN Workshop on Model Checking Software (SPIN), Lecture Notes in Computer Science 2648, Springer-Verlag, (2003) 235-239
6. Holzmann, G. J.: The SPIN Model Checker, Primer and Reference Manual. Addison-Wesley, (2003)
7. Holzmann, G. J.: The Model Checker SPIN. IEEE Transactions on software engineering, vol.23, no.5, (1997)
8. Martens, A.: Analyzing web service based business processes. In Proceeding of International Conference on Fundamental Approaches to Software Engineering, Part of the European Joint Conferences on Theory and Practice of Software, Lecture Notes in Computer Science vol. 3442, Springer-Verlag, (2005)
9. Leymann, F.: Web Services Flow Language (WSFL 1.0). IBM Corporation, May 2001.
10. Business Process Execution Language for Web Services (BPEL), Version 1.1, http://www-128.ibm.com/developerworks/library/ specification/ws-bpel. (2002)
11. Thatte, S.: XLANG: Web Services For Business Process Design, Microsoft Corporation, (2001), (http://www.gotdotnet.com/team/xml wsspecs/xlang-c/default.htm)

12. Nakajima, S.: Verification of Web service flows with model-checking techniques, presented at First International Symposium on Cyber Worlds, (2002)
13. Nakajima, S.: Model-Checking Behavioral Specification of BPEL Applications. Electronic Notes in Theoretical Computer Science 151 (2006) 89105
14. Bao, F., Zhang, L.: A Model Checking Method to Verify BPEL4People Processes. on The IEEE Symposium Advanced Management of Information for Globalized Enterprises. (2008)
15. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to Petri nets. Business Process Management, LNCS, vol. 3649, (2005) 220235.
16. Lohmann, N., Massuthe, P., Stahl, C. and Weinberg, D.: Analyzing interacting WS-BPEL processes using flexible model generation, Data and Knowledge Engineering 64 (2008) 3854
17. Massuthe, P., Reisig, W., Schmidt, K.: An operating guideline approach to the SOA, Annals of Mathematics, Computing and Teleinformatics 1 (3) (2005) 3543
18. van der Aalst, V.M.P.: Structural characterization of sound workflow nets, Computing Science Report 96/23, Eindhoven University of Technology, (1996)