International Conference on Control, Engineering & Information Technology (CEIT'13)
Proceedings Engineering & Technology - Vol.2, pp. 140-145, 2013
Copyright - IPCO

Modeling and formal verification framework of Web services composition

Rawand Guerfel *, Zohra Sbaï *, and Kamel Barkaoui †
*Université de Tunis El Manar,
Ecole Nationale d'Ingénieurs de Tunis,
SysCom Laboratory, Tunisia
zohra.sbai@enit.rnu.tn
†Conservatoire National des Arts et Métiers,
CEDRIC Laboratory, Paris, France
barkaoui@cnam.fr

***Abstract*—To ensure reliable and efficient communication in B2B environments, we relied on the use of composite Web services. Indeed, in some cases, a service cannot provide the functionality required by the user unless it communicates with other services. This leads to the notion of Web services composition. The communication of these Web services has to be guaranteed without errors such as deadlocks. In this way, formal verification of Web services composition is a focused-on research field. It is in this context that our work is proposed. We have developed a tool for modeling and formal verification of Web services composition which is based on the modeling of these services by open workflow nets: a special class of Petri nets used to model business processes which communicate together via interface places. This tool checks temporal properties written in CTL against a WSC model translated into SMV code. The verification is ensured by invoking the NuSMV Model Checker.**

***Keywords*—Model checking, Web Services Composition, open workflow nets, NuSMV, CTL, soundness property.**

I. INTRODUCTION

Web services have become widely used in the B2B environment to ensure the information exchange and sharing. They allow publishing Web services in the Web and therefore industries can have access to and benefit from them. Services can be defined by three standards: SOAP (which allows the exchange of XML messages), UDDI (to establish the list of available services) and WSDL (which allows the services description). But they suffer from some limitations. Indeed, in some cases, a single service cannot meet the needs of the user, but to to get there, it must communicate with another service. It is for this reason that we refer to the composition of Web services. In order to realize this composition, several approaches have been proposed. The first one is an approach based on workflows. The researchers adopted several languages to implement this approach, namely eFlow [5]. The second approach relied on XML. The most popular languages that are based on this approach are XLANG and BPEL4WS [6]. Unfortunately, these methods suffer from a total lack of semantic representations of services, which has led researchers to propose another approach based on ontology such as OWL-S [10]. It enables the exchange of information in a meaningful way. However, these approaches cannot verify certain properties such as QoS, accuracy, etc. This is why the description of the composition based on formal methods is necessary and interesting since these methods allow the analysis of Web services composition.

The composition of Web services faces many constraints and requirements. Because bugs can cause unexpected shutdown of the software, developers require reliable design of such complex systems. Many other developers require a warranty for the validity of use of these systems. They also require a total absence of undesired behavior to avoid damage that may be the result of error occurrence and which eventually cause serious financial losses. Mainly, we must validate these systems, before putting them up, with checking the sequence of the process and its operation. To perform this checking, two formal methods have been proposed: theorem proving and Model checking. With the first method, the system is seen as a theorem that we seek to prove from a set of axioms. But the model checking studies all possible cases of a computer system and eventually presents a counter example in case of error. Since the identification of the error in the process became a critical need, we chose the formal verification based on model checking. This process occurs in three different phases: Modeling of the system behavior, Specification of the requirements to be verified and finally the Verification. Two cases arise at this latter stage: either the system is validated or not and in the second case, the model checker displays a counter example, which may be used to specify the exact location of the error.

In this context, we propose a tool to model and formally verify the composition of Web services. This tool is characterized by its graphical environment, facilitating its use, and in which the user can first model graphically the overall process by means of open workflow nets. Then, the modeled process is generated in SMV language, the input language of NuSMV model checker. Finally the tool can check soundness properties, specified in CTL, and above all display a counter example in case there are problems in phase of process modeling.

The rest of this paper is organized as follows. Section 2 is dedicated to present preliminaries on Web services composition (WSC), Petri nets as well as model checking techniques.

We present in section 3 our approach of formal verification of WSC. In section 4, we describe the features of our tool implementing and facilitating the formal verification of WSC. Section 5 draws a case study in order to illustrate the approach. In section 6, we review some related work. Section 7 concludes the paper and announces directions to future work.

II. Preliminaries

A. Web services composition

Web services composition (WSC) consists on the construction of new Web services by composing existing ones. A composite service is a new service with added value obtained by combining other existing services. It can be the allocation of basic services or composite services. Figure 1 shows the obtaining composite service.

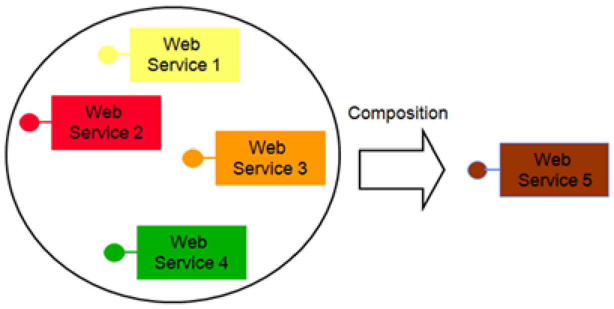


Fig. 1. Schematic description of WSC

B. Petri nets

Petri nets were originally developed to meet the need in specifying process synchronization, asynchronous events, concurrent operations, and conflicts or resource sharing for a variety of industrial automated systems at the discrete-event level. For that reasons, we propose to model WSC using Petri nets [3].

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.

C. Model checking Techniques

Formal methods and tools have proved useful to give high-level and precise descriptions of computer systems, and to analyze exhaustively these systems at early phases of the system development process. They are a particular kind of mathematically-based techniques for the specification, development and verification of software and hardware systems. The use of formal methods for software and hardware design is motivated by the expectation that, as in other engineering disciplines, performing appropriate mathematical analysis can contribute to the reliability and robustness of a design.

Model checking is a formal verification method 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 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 [1], BLAST [12] and SPIN [9]. NuSMV is an extension and re-implementation of SMV symbolic model checker, the first model checking tool which is based on Binary Decision Diagrams (BDDs) [4]. The tool has been designed to be an open architecture for model checking. It has been developed in a joint project between ITC-IRST, the University of Genoa, the University of Trento and Carnegie Mellon University.

NuSMV allows the formal verification of finite state systems based. This tool takes as input a model described in the SMV language and supports the verification of temporal properties written in CTL and LTL. It offers many features including essentially the navigation of a counter example. In fact, all the model checkers can generate a counter example, the particularity of NuSMV is that it offers the possibility to the user to move from one state to another and even to assess the CTL expression of the counter example status. [1]

CTL (Computation Tree Logic) is a temporal logic which permits to express certain properties from a given state graph in order to check the validity of these properties. CTL formulas are defined by the following grammar:

$$\begin{aligned}\varphi \ ::= \ & \top|\bot|p \\ & |\neg\varphi|\varphi \wedge \varphi|\varphi \vee \varphi|\varphi \rightarrow \varphi \\ & |AX\varphi|EX\varphi|AF\varphi|EF\varphi|AG\varphi \\ & |EG\varphi|A[\varphi \sqcup \varphi]|E[\varphi \sqcup \varphi]\end{aligned}$$

Where A models all possible paths from the current time, E models at least one path from the current moment A and E are usually followed by one of the following prefixes: X, U, G and F.

A CTL formula f is built from atomic propositions corresponding to variables, boolean operators such as $\neg$, $\vee$, $\wedge$, $\rightarrow$, and temporal operators. Each temporal operator consists of two parts. A path quantifier (A or G) followed by a temporal modality (F, G, X, U). The temporal operators are interpreted relative to a current state s. The path quantifier indicates if the temporal operator expresses a property that should hold on all paths starting at s (denoted by the universal path quantifier A), or at least on one of these paths (denoted by the existential path quantifier E). The temporal modalities are interpreted as follows:

- Xp (neXt time p) is true if the formula p is true in the state reached immediately after the current state in the path.
- Fp (Future p) is true if there exists a state in the path where the formula p is true.
- Gp (Globally p) is true if p is true at every state in the path.
- pUq (p Until q) is true if there is a state in the path where q is true and p is true in all preceding states (if any). This definition is the so called "strong until". "Weak until" would also be true when q holds forever.

III. WSC FORMAL VERIFICATION APPROACH

Our goal is to achieve a formal verification of WSC based on NuSMV model checker. To ensure the invocation of NuSMV for model checking, we have to prepare the model of the WSC in SMV code and to specify the requirements to be verified in CTL or LTL temporal logic.

Due to the difficulty of describing the system in SMV language on the one hand and the expressive power as well as the graphical nature of Petri nets on the other hand, we propose to model first the WSC in Petri nets and then to translate the obtained model into SMV. Afterwards, we propose to specify in CTL the requirements to be verified. Finally, we invoke NuSMV to verify if the model meets the requirements. This approach is presented in figure 2.

The first phase consists on modeling the WSC based on Petri nets. We, especially, used open workflow nets to model Web services which communicate with other services. Open workflow nets (oWF-nets) result from the extension of workflow nets (WF-nets) by adding external places for the communication of the external environment of the considered Web services. Now for WF-nets, they result from the application of Petri nets to workflow management [13]. A WF-net is a Petri net with two special places: input place i containing initially x tokens (x is the number of instances ready for execution) and a final place f. In a WF-net, each node is in a path from i to f. For WSC, we model each Web service by an oWF-net describing the control flow between the tasks ensured by this service and containing interface places used to communicate with other Web services. The overall obtained model is a Petri net with a set of input places, a set of output places and a set of interface places.

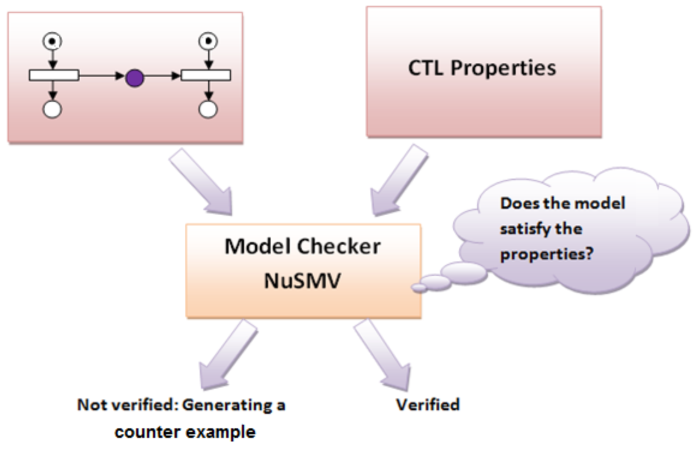

CTL Properties
Model Checker NuSMV
Does the model satisfy the properties?
Not verified: Generating a counter example
Verified

Fig. 2. The WSC formal verification approach

The second phase is the translation of the obtained Petri net into an SMV code. We ensured this translation after proposing a method to describe a WSC Petri net in the SMV language. The SMV code starts by declaring the header MODULE. In this module, we find the definition of:

- VAR: to declare variables (places and transitions).
- INIT: to initialize variables.
- TRANS: the transition relation defining the evolution of the various states.
- DEFINE: to define the eventual used macros.

In the VAR clause, we define s arrays of places to save the places marking, s arrays of transitions used to save the firing of the transitions (s is the number of Web services involved in the composition) and an array used to save the marking of interface places. These arrays are initialized in the INIT clause as follows: the places cells are initialized to 0 except those corresponding to input places; they are initialized to the number of tokens entered at the beginning of the modeling; the interface places cells and those of transitions are also initialized to 0.

The dynamic behavior of the system is described in the TRANS part in which is described the evolution of the system states. This description is ensured via macros defined in the DEFINE part. The evolution of a system state is ensured if some firing condition is verified, i.e. if each input place of a given transition contains at least one token. If this condition is verified, the firing is saved by increasing by 1 the transition cell, increasing by 1 the cell corresponding to each output place of the transition and by decreasing by 1 the marking of each input place.

Once the model is specified in SMV, we have to specify in CTL any requirements to be checked by NuSMV model checker. We propose here to present and formulate soundness properties of WSC. The soundness property is first introduced in [14] to verify the correct execution of workflow processes. The soundness of a WSC guarantees the absence of livelocks, deadlocks (blocking) and other anomalies that can damage the system in question. It requires that the overall Petri net has a clean termination and that at any state of the process, there is at least one task which can be executed. Formally, a composed oWF-net os s services is called sound if and only if the following three requirements are satisfied:

1) Termination: For each reachable marking (from $M_0 = [i_1, i_2, .., i_s]$) the final marking $M_f = [f_1, f_2, .., f_s]$ is reachable;
2) Proper Termination: For each reachable marking M, if $M \geq M_f$ holds then $M = M_f = [f_1, f_2, .., f_s]$;
3) Absence of deadlocks: Starting with a token in each input place i_j, it should be possible to execute a random task by following the appropriate route through the composite oWF-nets, i.e. there are no dead transitions. $\forall t, \exists M \in [M_0\rangle$: $M[t\rangle$.

In CTL, this soundness property is specified by these formulas:

1) $EF(\underset{s=1}{\overset{s=n}{\&}}(PLS_s[f_s] > 0))$
2) $EG((\underset{s=1}{\overset{s=n}{\&}}(PLS_s[f_s] > 0)) \rightarrow (AF(\underset{s=1}{\overset{s=n}{\&}}(PLS_s[f_s] > 0 \;\&\; PLS_s[\backslash f_s] = 0) \;\&\; \underset{k=1}{\overset{k=ni}{\&}}(I[p_k] = 0))))$
3) $AG(EG(\underset{s=1}{\overset{s=n}{|}} TRS_s[t] = 1))$

Where n is the number of involved services, ni is the number of interface places, $PLS_s[f_s]$ denotes the marking of the final place f_s of the service s, $PLS_s[\backslash f_s]$ denotes the marking of any place except f_s, I is the table saving the marking of interface places and TRS_s the table saving the firing of transitions of service s.

IV. Implementation of the WSC Formal Verification

In order to facilitate to users the WSC analysis, we proposed to fully implement this approach. Hence,we developed in Java a graphical tool which allows the user to (1) model a Web service which may be simple or composite by respectively workflow nets [14] or open workflow nets, (2) compile it by verifying if the constructs used are seemly (3) generate the corresponding SMV code and (4) verify soundness properties.

In more details, the tool features are :

- Modeling Web services composition by giving the user the possibility to draw the oWF-net based model of the WSC. To do this, it was suggested to the user to enter as many input places and transitions as he wants, so he can model each Web service by a WF-net. Then, he adds interface places, which are distinguished from other places, to interconnect the different Web services. After this modeling, and before doing other tasks, the user must compile the overall process in order to check that the model presented is syntactically correct. In case of error, a message will be displayed and the following tasks won't be enabled.
- Request of an SMV code generation: Here, if an edited model is compiled successfully, the user can request to show the corresponding SMV code.
- Checking of the model: The system can not verify the user model unless it is compiled successfully. Here, the tool will check the soundness of the modeled process.
- Verification of a property: If the user knows CTL, he can enter directly some requirements to verify in a CTL formula and check it. To do this, he is given a CTL editor to facilitate the formulas entering and to prevent syntactical errors.

To better present these different features, we propose to draw a UML sequence diagram, illustrated in Figure 3, which details the interactions between the user, the system and the model checker.

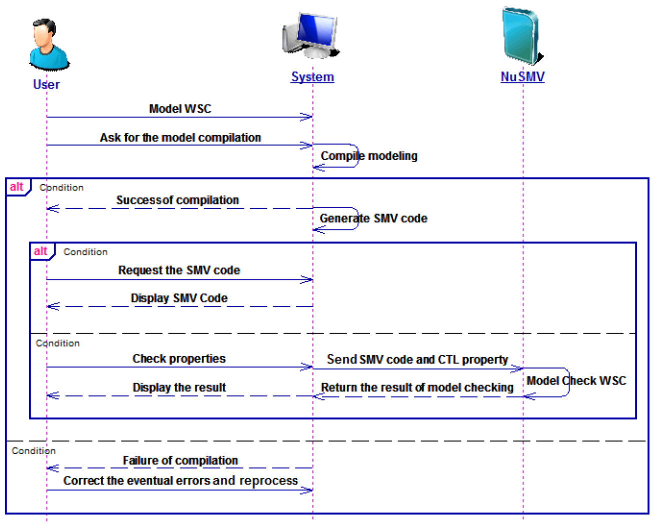


Fig. 3. Sequence diagram describing the general feature of the tool

In order to achieve the implementation of this tool in Java, we divided our application into five packages:

- Package PetriNet: It contains the different classes defining the objects that the user will enter: Input place, Output place, Transition, interface Place, Arc, Place and the class that defines the process.
- Package Library: contains all the libraries needed for the application.
- Package Interface: It is responsible for the Graphical Interface tool. It allows the user to model the composite oWF-Net by giving him a hand to draw his process. It also provides features such as side window and display management.
- Package Parser: It is responsible for the syntactical checking of the WSC. It also ensures the generation of an XML description file of the WSC compliant to the PNML norm.
- Package Model Checking: It permits to generate the SMV code and thus to validate the overall process edited by the user by invoking the NuSMV model checker.

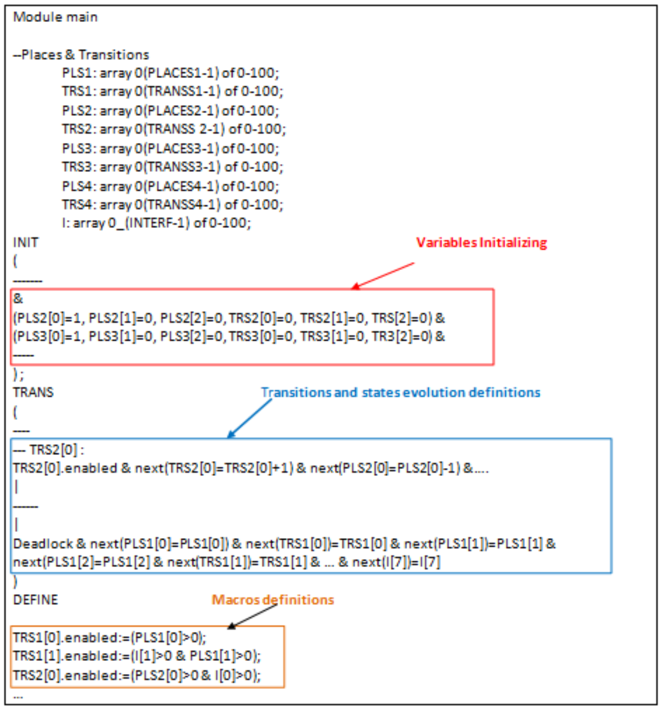
V. Case study

To illustrate the approach presented and to test the implemented functioning of the tool, we propose to study a Web portal of credit management. We treat only the case of a credit request by a client. The scenario is as follows: The client begins by entering his id (name and password). A service is needed to check the validity of this id. If it is not valid, then the customer must enter his id again. Otherwise he can request a credit. Then, he enters his guarantee and balance. At this stage, the validation service intervenes to check the availability of this balance. If it is not available, the process ends, else, the service suits the customer's request. Here, the processing request service puts it in the list of tasks to be performed. A supervisor withdraws a request from this list and evaluates its private part (by checking the relevant information to the client) and its public part. In assessing the public part, the supervisor may either:

- Reject the request: then the supervisor sends a rejection message to the customer.
- Update the application: in this case, the supervisor sends a message to the client to ask him to update his application. This client can either accept or reject this proposal.
- Accept the request: in this case, a supervisor updates and evaluates the request. Then, he sends the customer an offer that he can accept or reject.

Modeled by oWF-nets, this portal is described in figure 4.

The SMV code is generated by our tool in order to be checked by NuSMV. A piece of the generated code is given in figure 5.



```
Module main

--Places & Transitions
        PLS1: array 0(PLACES1-1) of 0-100;
        TRS1: array 0(TRANSS1-1) of 0-100;
        PLS2: array 0(PLACES2-1) of 0-100;
        TRS2: array 0(TRANSS 2-1) of 0-100;
        PLS3: array 0(PLACES3-1) of 0-100;
        TRS3: array 0(TRANSS3-1) of 0-100;
        PLS4: array 0(PLACES4-1) of 0-100;
        TRS4: array 0(TRANSS4-1) of 0-100;
        I: array 0_(INTERF-1) of 0-100;
INIT
(
-------
&
(PLS2[0]=1, PLS2[1]=0, PLS2[2]=0, TRS2[0]=0, TRS2[1]=0, TRS[2]=0) &
(PLS3[0]=1, PLS3[1]=0, PLS3[2]=0, TRS3[0]=0, TRS3[1]=0, TR3[2]=0) &
-----
);
TRANS
(
----
--- TRS2[0] :
TRS2[0].enabled & next(TRS2[0]=TRS2[0]+1) & next(PLS2[0]=PLS2[0]-1) &....
|
------
|
Deadlock & next(PLS1[0]=PLS1[0]) & next(TRS1[0])=TRS1[0] & next(PLS1[1])=PLS1[1] &
next(PLS1[2]=PLS1[2] & next(TRS1[1])=TRS1[1] & ... & next(I[7])=I[7]
)
DEFINE

TRS1[0].enabled:=(PLS1[0]>0);
TRS1[1].enabled:=(I[1]>0 & PLS1[1]>0);
TRS2[0].enabled:=(PLS2[0]>0 & I[0]>0);
...
```

Fig. 5. A snapshot of the generated SMV code of the WSC studied

VI. Related Work

WSC is a very important topic which attracted the attention of researchers and industrials. However, we still note a lack in the number of WSC verification tools. We review in this section different works which highlighted the WSC in general and their verification in particular. Fu et al. [15] modeled the composition of Web services by finite state automata. Their model assumes that the communication is done between pairs of services through asynchronous messages. At first, They specified services in BPEL. Then they modeled them by the finite state automata in order to specify them with Promela language and eventually check them with SPIN.

Ouyang et al. [2] studied the verification of the Web services composition that are based on BPEL. To do so, they first made the mapping of services described in BPEL to those described by Petri nets using the tool BPEL2PNML. This tool takes as input the code written in BPEL and displays the output to a file with a PNML syntax. Receiving the file, the WofBPEL tool generates an XML file containing the results of the analysis.

Narayanan et al. proposed in [11] an interpreter DAML-S, based on the environment KarmaSIM to ensure the description, simulation, and verification of the composition of semantic Web services. In fact, they focused their work on ontological Web services modeled by Petri nets and based on DAML-S.

Foster et al. [7] have implemented a plugin LTSAWS to the existing tool LTSA, which uses FSP to describe the behavior of models. The tool allows integrating different specifications and implementations of Web service composition and audits.

Our work is not limited to a simple checking of the composition. In fact, the tool we proposed permits to model the composition by oWF-nets offering hence a very understandable graphical environment which is easy to use. Then, it generates a SMV code that is given as an input to NuSMV, which will eventually check the soundness property.

VII. Conclusion

We proposed in this paper a tool for formal verification of Web services composition. These services are modeled by oWF-nets composed of a set of workflow nets interconnected via interface places. These places ensure the communication between the different services. This model is generated to an SMV code that is sent to NuSMV Model Checker. NuSMV verifies certain properties formulated in CTL such as soundness properties.

In case of failure, NuSMV generates a counter example that we analyse and present to the user in order to help him to find the error and then correct the process. This tool ensures more safety in the design of complex systems and therefore avoids the technical and financial losses that may result from small errors. More precisely, we benefited from model checking as a powerful technique to verify and to detect the eventual errors at an early stage.

The presented tool can be extended by the integration of temporal constraints when modeling the composition of Web services. These constraints allow us to specify a certain time to respect for each Web service operation.

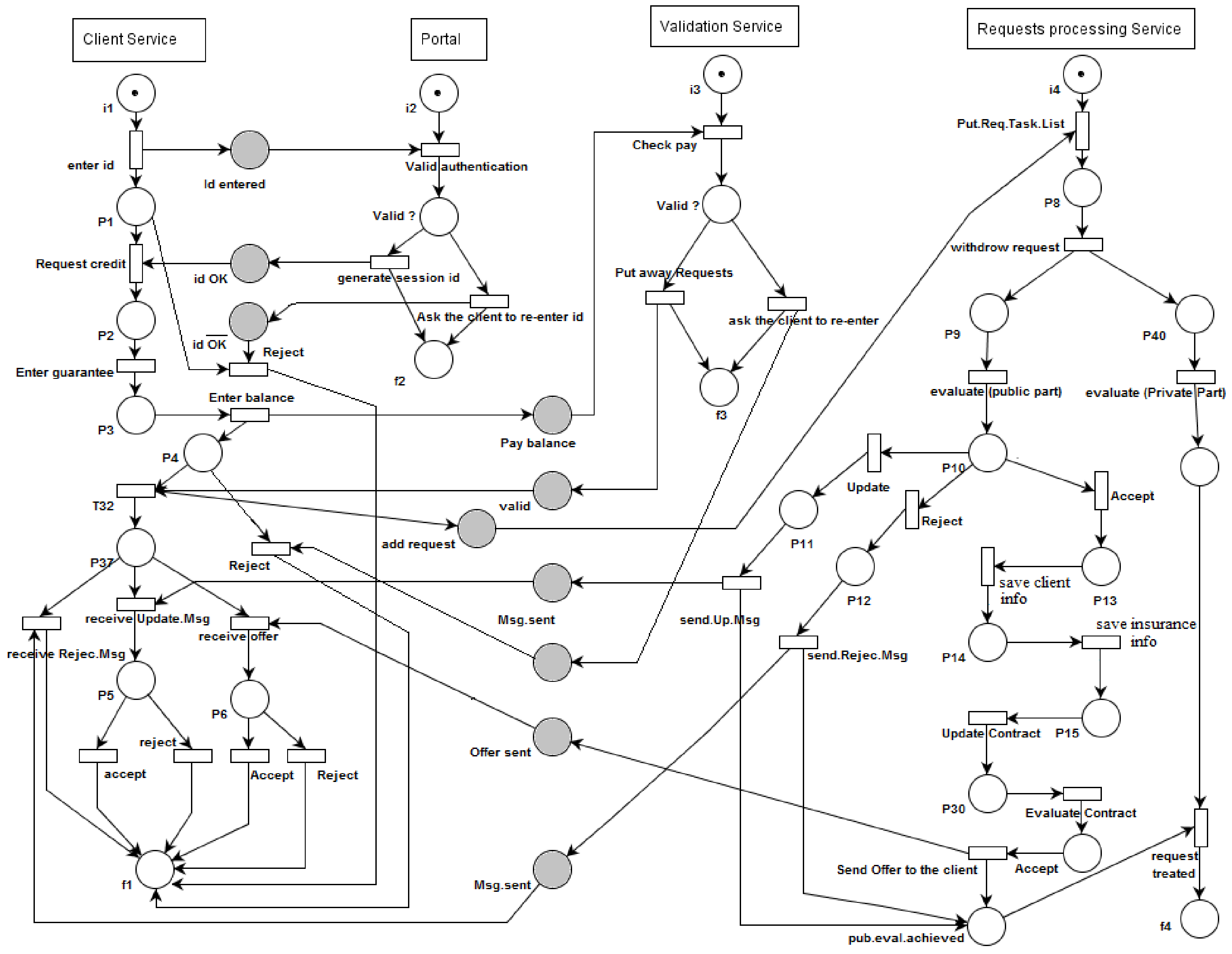


Fig. 4. The credit management Petri net based model

References

[1] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella. NuSMV 2: An Open Source Tool for Symbolic Model Checking. In Proceeding of International Conference on Computer-Aided Verification, 2002.

[2] C. Ouyang, E. Verbeek, W.M.P. van der Aalst, S. Breutel, M. Dumas and A.H.M. ter Hofstede. Formal Semantics and Analysis of Control Flow in WS-BPEL. Science of Computer Programming archive, Vol. 67, pp. 162–198, 2007.

[3] K. Barkaoui, R. Ben Ayed and Z. Sbaï. Workflow Soundness Verification based on Structure Theory of Petri Nets. International Journal of Computing and Information Sciences (IJCIS), Vol. 5(1), pp. 51–61, 2007.

[4] K.L. McMillan. Symbolic model checking. In Kluwer Academic Publ., 1993.

[5] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, M-C. Shan. Adaptive and Dynamic Service Composition in eFlow. CAISE, 2000.

[6] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, S. Weerawarana. Business Process Execution Language for Web Services, version 1.0. Standards proposed by BEA Systems, International Business Machines Corporation, and Microsoft Corpora-tion, 2002.

[7] H. Foster, S. Uchitel, J. Magee, J. Kramer. LTSA-WS: A Tool for Model-Based Verification of Web Service Compositions and Choreography. In Proceeding of the 28th international conference on Software Engineering (ICSE), 2006.

[8] G. J. Holzmann. The SPIN Model Checker, Primer and Reference Manual. Addison-Wesley, 2003.

[9] G. J. Holzmann. The Model Checker SPIN. IEEE Transactions on software engineering, vol.23, no.5, 1997.

[10] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, L. A. Stein. Owl web ontology language reference, http://www.w3.org/TR/owl-ref/. 2004.

[11] S. Narayanan and S.A. McIlraith. Simulation, Verification and Automated Composition of Web Services. Proceedings of the 11th international conference on World Wide Web, pp. 77–88, 2002.

[12] T.A. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Software Verification with Blast. In Proceedings of the 10th SPIN Workshop on Model Checking Software (SPIN), Lecture Notes in Computer Science 2648, Springer-Verlag, pages 235-239, 2003.

[13] W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, vol. 8, pp. 21-66, 1998.

[14] W.M.P. van der Aalst. Verification of Workflow nets. ICATPN 97, LNCS, vol. 1248, 1997.

[15] X. Fu, T. Bultan and J. Su. Analysis of Interacting BPEL Web Services. Proceedings of the international conference on World Wide Web'04, 2004.