Formal Verification of Web Service Orchestration Using Colored Petri Net

C. Dechsupa, W. Vatanawood, and A. Thongtak

Abstract—**Composite web service is typically comprised of a main orchestration description written in BPEL and its associated set of simple and individual web services. A common way to test the composite web service is to execute the BPEL along with its fully implemented version of the individual web services. Unfortunately, a particular web services orchestration in BPEL is difficult to be tested beforehand alone, without the availability of the related web services. In this paper, we propose an alternative mean to verify the web services orchestration in the early stage of the high level design process using the formal verification of the BPEL and its imitated stubs, called dummy web services. In our approach, a relevant dummy web service is generated to cope with its defined WSDL and the equivalence class partitioning technique is specifically focused. Thus, the valid and invalid classes of the invocation of the web services are simulated. The simple transformation rules of the BPEL alone, without the detailed of its associated individual web services, into Colored Petri Net are proposed. The resulting Colored Petri Net of a BPEL is correctly and consistently transformed and verifiable in CPN Tool using the equivalence class partitioning technique.**

Index Terms—**Formal Verification, Dummy Service, Composite Web Service**

I. Introduction

PARADIGM for distributed computing, Service-Oriented Computing (SOC)**[1]** provides a manner to create new application architecture for cooperating services loosely connected, creating dynamic business processes and agile applications. The SOC promise requires the design of Service-Oriented Architectures (SOAs) that allow the development of simpler and cheaper distributed applications. Web Service Composite Application Framework (WS-CAF) [2] is an open framework, generic framework for application that contains multiple services used together. It triggers an emerging of various software tools used to describe structural and behavioral of web service application. Composite web service applications are established from several individual web services. Generally, these applications have a lot of sub-processes, data paths, input vectors, and state transitions that affect application complexity.

The standard for assembling a set of discrete services, the Web Services Business Process Execution Language (WS-BPEL) is used to model the behavior of both executable and abstract processes. All paths of service component architecture need to be aware of the business process, operations to execute, messages to exchange, and the timing of message exchanges. To ensure design, formal verification is the methodology for model checking that is the algorithmic analysis of programs to prove the properties of their executions.

Model checking tools are distinguished formal verification methods that can be ensured the correctness of application. They are used to detect and diagnose the faults in software and hardware. Currently, various verification tools have been used to verify a model, which they support automatic checking.

We propose methods for model abstraction and verification of composite web service application described as WS-BPEL. The WS-BPEL would be transformed into Colored Petri-Net model and we also introduce the dummy web service generation technique. The Colored Petri-Net tool is used to draw the resulting model and check the desired properties. The remaining of this paper is organized as follows: Section II describes background for web service verification. Section III reviews the related researches. Section IV discusses the proposed approach and Section V illustrates our implementation with a simple case study. Section VI is our conclusion.

II. Background

A. Web Service Business Process Execution Language

Business Process Execution Language is commonly known as BPEL or WS-BPEL [3], which is used for business processes definition as coordinated sets of Web service interactions to achieve business goals. It uses an XML-based language that supports the web services technology stack, including SOAP, WSDL, UDDI, WS-Reliable Messaging, WS-Addressing, WS-Coordination, and WS-Transaction , these are used to define process definition, model including the grammar for describing the application behavior based on interactions between the process and its partners.

In order to invoke the service partner, Web Services Description Language (WSDL) [4] is a standard format for describing a web services interface that specifies the

C. Dechsupa is a Ph.D. student at department of computer engineering faculty of engineering, Chulalongkorn University, Bangkok, 10330 Thailand (e-mail: chanon.de@student.chula.ac.th).

W. Vatanawood is an associate professor at department of computer engineering faculty of engineering, Chulalongkorn University, Bangkok, 10330, Thailand (to provide phone: (+66) 0-2218-6956-7; fax: (+66) 0-2218-6955; e-mail: wiwat@chula.ac.th).

A. Thongtak is an assistant professor at department of computer engineering faculty of engineering, Chulalongkorn University, Bangkok, 10330, Thailand (e-mail: arthit.t@chula.ac.th).

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

location of the service, and the names of the methods of the service. BPEL introduces a mechanism to define how individual or composite activities within a process. Actually, BPEL only requires WSDL port types, and those port types map to their partner links in the BPEL process. How those partner links are connected to physical services defined in a WSDL document.

B. Dummy Web Service.

Dummy Web Service or Mockup Service is the rapid web service prototyping that performs the static mock implementation, adding the functionality of the desired operations and the creation of the alternative responses. It allows an implementing and a client testing that is no need to wait for the completely actual service or to avoid the directed service invocation during the testing phase. For instance, The SOAP Service Mocking functionality in *soapUI* [5] is called a "*MockService*". It is used to simulate the number of WSDL contracts; to build the scripting functionality and to simulate basically any kind of the desired behavior that roughly is programed by the designer. Although the developers can use WSDL for obtaining the interface of service, they cannot catch its original implementation. Therefore, the dummy service should be built to resemble the original service as possible.

C. Colored Petri-Net (CPN)

Colored Petri-Net or CPN [6] is the graphical language for constructing models of concurrent systems and analyzing their properties. CPN is extended classical Petri-net with data, hierarchy and time; an attached data value is called a token color. Color set is a place that contains token. CPN provides the primitives for the description of the synchronization of the concurrent processes, while CPN ML programming language [7] provides the primitives for the definition of data types and the manipulation of data values. The CPN allows the designer to focus on the problem models using CPN Tool [8] for editing, simulating, and analyzing Colored Petri nets model.

III. Literature Review

Currently, many verification techniques have been used to verify composite service. Researches in web service composition mainly focus on formal verification of service composition based on interaction activities from basic activity in BPEL [9, 10]. Artifacts in service oriented system described with formal mathematical model allowed the structural and behavioral definition. Additionally, many verification tools support the automatically correctness checking of an abstract model. An example of model correctness checking approach is the formal verification with CPN tool, which the application is written in CPN model and temporal properties are written in Computational Tree Logic (CTL) to verify its properties.

Wei Tan et al. [11] provided an approach to analyze an interservice compatibility and automatically compose services. The service constraints were written in BPEL. They transformed BPEL description to CPN model and adopt mediation to make their compatible without changing their internal logic, and then analyzed their composition with the message passing of the mediator. Likewise, Yingmin LI et al. [12] addressed the diagnosis of faulty data and the faulty activities of orchestrated web services. They proposed the inequations solving algorithm that helps to improve the fault detection. They defined the color propagation functions for each data dependency relation. Mapping each primitive data to a place and each basic activity to a transition are the mapping technique that was used for transforming BPEL to CPN model, it was verified by model-based diagnosis framework.

Formal verification technique has to depict all of the behaviors of application. If an individual web service is the service provided by an outside service provider, the designers could only catch its interface. Dummy service generation is the alternative mean to imitate the original service. It can be used for development and web service testing in the early stage of the high level design process. Sylvain Hall [13] proposed an approach for generating dummy service which was input-output patterns imitation. A dummy service can be generated from a set of LTL-FO+ formula expressing the wide range of constraints, including message sequences, parameter values, and interdependencies.

In a similar way, our methods are defining simple rules of the transformation for BPEL to create CPN model and proposing the generating concept of the dummy web service to abstract the implementation of individual web services, the equivalence class partitioning technique is particularly focused. Both techniques can be used understandably and effectively.

IV. Our Approach

The overview of our formal verification process is shown in Fig. 1. The target BPEL file is expected and its BPEL elements are extracted and considered. The BPEL elements represent all of the possible activities, divided into two classes: basic and structured. Basic activities describe elemental steps of the process behavior, such as *Invoke*, *Assign*, *Receive*, *Reply*, *Wait*, etc. While, structured activities encode control flow logic, such as *Choice*, *For Each*, *Parallel*, *Sequence*, *While loop*, *Repeat until loop*, etc. The type of each BPEL element would be analyzed and mapped into CPN element relevantly, according to our proposed transformation rules. An *Invoke* or *Reply* activity would be specifically considered with its associated individual web service interface, shown as WSDL interface file. A dummy service would be generated to each *Invoke* or *Reply* activity in term of CPN elements as well.

A. Transform BPEL to CPN.

In this section, the definitions and one simple rules of transformation of BPEL into CPN model are proposed.

Definition 1: BPEL model.
A BPEL model is a n-tuple BP= (*BAct*, *SAct*, *Plnk*, *Var*) where

BAct is a set of basic activities {*Invoke*, *Assign*, *Receive*, *Reply*, *Wait*, *Validate* ...}.
SAct is a set of structured activities {*If-Else*, *Choice*, *For Each*, *Parallel*, *Sequence*, *While loop*, *Repeat until loop*,.}
Plnk is a set of partner links.
Var is a set of variables in BP.

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

Definition 2: CPN model.
A CPN model is a n-tuple CP = (*PL*, *TN*, *Arc*, *Colr*, *PLColr*, *Grd*, *VV*)
where
PL is a set of places.
TN is a set of transition.
Arc is a set of arcs (*x*,*y*) where (*x*,*y*) ∈ (*PL* x *TN*) ∪ (*TN* x *PL*)
Colr is a set of color sets defined in BP.
PLColr is a color set function *PL* → *Colr*.
Grd is a guard function *T* → *Gexp* where *Gexp* is a predicate guard expression, evaluated to Boolean True or False.
VV is a set of variables in CP.

According to the definitions of BPEL and CPN models, we define the simple rules of transformating BPEL into CPN.

Transformation Rules from BP to CP:

1) For the set of the consecutive basic activities of "*Receive*" and "*Assign*", a place in *PL* is defined in CP.
2) For a basic activity, other than "*Receive*" and "*Assign*", a transition in *TN* is defined in CP.
3) For a partner link in *Plnk*, an additional dummy web service is defined.
4) For structured activity of "*If-Else*", a transition in *TN* are defined in CP.
5) For each pair of activities in *BAct* ∪ *SAct*, a buffer place in *PL* is defined in CP.

6) For each variable in *Var*, a corresponding variable in *VV* is defined in CP.
7) For each type of variable in *Var*, a corresponding *Colr* is defined in CP.

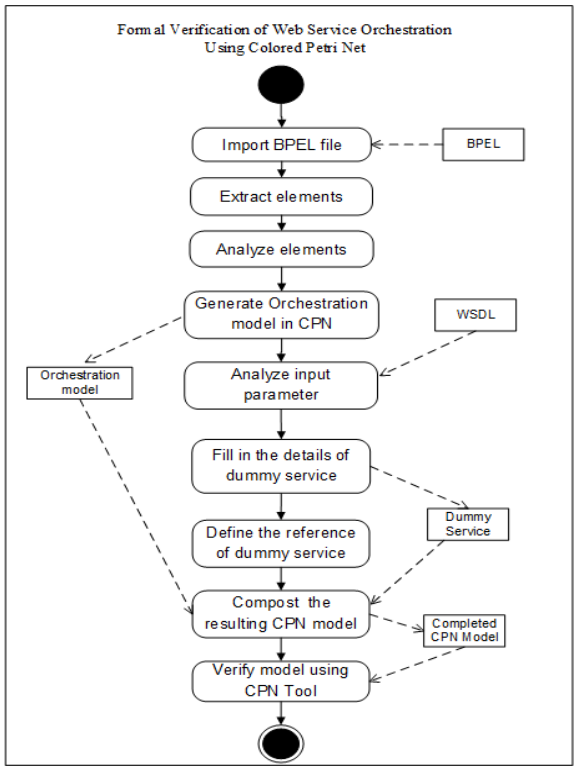
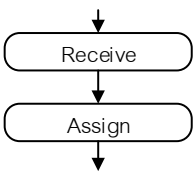
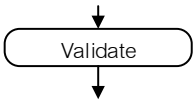
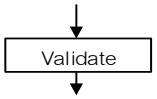
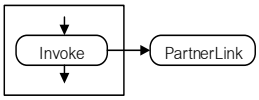
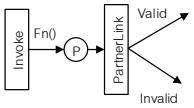
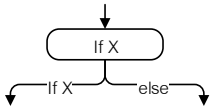
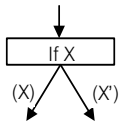
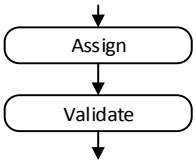
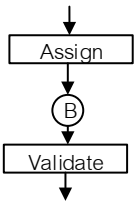


Fig. 1. Formal verification of web service orchestration.

Some examples of BPEL to CPN elemental transformation in terms of notations, are shown in Table 1. Moreover, a sample of dummy web service defined for a particular partner link would be described in the next section.

TABLE I
TRANSFORMATION RULES AND THEIR EXAMPLES

Rule#	BPEL	CPN
1)		
2)		
3)		
4)		
5)		
6)	Types declaration in BPEL's XML	Colored sets in CPN
7)	Variables declaration in BPEL's XML	Variables sets in CPN

B. Dummy Web Service Generation

The implementations of dummy web service are imitatively generated from the WSDL of individual service. The dummy web service generation consists of two steps.

1). Analyze the interfaces of the dummy web service those are defined in *interface* tag and *types* tag. This step produces the dummy web service name, the input parameters, the data type of input parameters, the boundary value of each input parameters, and the data type of return value.

2). Fill in the details of the dummy web service those are derived from the first step.

An equivalence class partitioning technique is used for parameters analysis of the details of the dummy web service. Each input parameter is used for analyzing the range of equivalence class. This research focuses on six data types of inputs and outputs: integer, double, decimal, byte, string and enumeration. In case of integer or double or decimal, the equivalence classes of input parameter are partitioned into three classes which are the valid class, the invalid class of lower-bound and the invalid class of upper-bound.

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

For instance, web service provider requires *@salary* parameter declared as decimal and the boundary values specified between 10 and 100, it means that the valid class is {*100>=@salary>=10* }, the invalid class of lower-bound is {*@salary<10*} and the invalid class of upper-bound is{*@salary*>100}. In case of string or enumeration, the equivalence class is partitioned into two classes those are the valid class comes from the default value or the choices of such parameter, and invalid classes are the values those are not in the default value or the choices of such parameter.

For instance, web service provider requires *@LoanType* parameter declared as enumeration. It consists of three choices: *EML*, *NML*, and *SPL*. It means that the valid class is {*@LoanType* in (EML, *NML*, *SPL*)} and invalid class is {*@LoanType* not in (EML, *NML*, *SPL*)}. Likewise, the outputs of service can only be considered in the valid class and single invalid class (compose the lower-bound and upper-bound to a class) in order to control the output values of dummy web service.

Fig. 2 illustrates the algorithm used. Line number 6-13 describe the input parameters those are defined as integer or double or decimal, line number 14-18 describe parameter which is defined as byte, line number 19-22 describe parameter defined as string, and the input parameter defined as enumeration describes in line number 23-28. The number of parameter per service operation is fixed in two parameters those are the limitation of this work.

```
1  AnalyzeParameter(inputList[]){
2  /*inputList[] is parameter list :parameter name, data type
3  ,min value, max value and set of choices (enumeration case) */
4  for j=0,j< = inputList.lenge, j++
5       nOfEqClass = the number of equivalence class
6       if inputList.[j] in [ 'integer', 'double', 'decimal']
7             set  nOfEqClass[j]=3
8             set ParaVal[0] = (max<=@par_value<=min )  -- Valid
9             set ParaVal[1] = (@par_value < min)    -- Invalid Lower
10            set ParaVal[2] = (@par_value>max)    -- Invalid Upper
11        end if
12       if inputList.[j] = 'byte' then
13            set  nOfEqClass[j]=1
14            set ParaVal[0] = @par_value in (0,1)  -- Valid
15       end if
16       if inputList.[j] = 'string' then
17            set  nOfEqClass [j]=2
18            set ParaVal[0] =@par_value = Default value  -- Valid
19            set ParaVal[1] =@par_value <> Default value -- Invalid
20       end if
21       if inputList.[j] = 'enumeration' then
22            set  nOfEqClass = 2
23            set ParaVal[0] =@par_value in (choices.value) -- Valid
24            set ParaVal[1] =@par_value not in (choices.value) -- Invalid
25       end if
26  end for
27      return value case  = nOfEqClass [0]* nOfEqClass [1]
28  }
```

Fig. 2. The algorithm for input parameter analysis.

An example of WSDL elements, Service provider provides the web service for the loan credit limit calculation. The interface name of service is *CalculateCreditLimit* and the operation name is *OpCheckLoanCredit*. The input parameters of this operation are *LoanType* declared as enumeration and *RequestAmt* declared as decimal and the return value of this service is *chkResponse* that is defined as double. The value of *LoanType* is defined in three choices (*EML*, *NML*, and *SPL*) and the value of *RequestAmt* is defined in range between 1,000 and 100,000. The ranges of *RequestAmt* are partitioned into three classes: valid class (100,000>=RequestAmt>=1,000), invalid class of lower-bound (RequestAmt<1,000) and invalid class of upper-bound (RequestAmt>100,000). The WSDL element of loan credit limit calculation demonstrates as Fig.3. The numbers of possible cases of dummy web service are six cases described in if-else guard conditions those are illustrated in Fig. 4.

```
……
<xs:simpleType name="loanCode">
  <xs:restriction base="xs:string">
    <xs:enumeration value="NML"/>
    <xs:enumeration value="SML"/>
  </xs:restriction>
</xs:simpleType>
<xs:simpleType name='amount'>
 <xs:restriction base='decimal'>
  <xs:minInclusive value="1000"/>
  <xs:maxInclusive value="100000"/>
 </xs:restriction>
</xs:simpleType>
<xs:complexType name="chkRequest">
<xs:sequence>
<xs:element  name="LoanType" type="tns:loanCode" />
<xs:element  name="RequestAmt" type="tns:amount" />
</xs:sequence>
</xs:complexType>
<xs:element name="chkResponse" type="xs:double"/>
<xs:element name="InvalErr" type="xs:string"/>
</xs:schema>
</types>
<interface  name = "CalculateCreditLimit" >
<fault name = "InvalErr" element = "ghns:InvalErr"/>
<operation name="opCheckLoanCredit".....>
<input messageLabel="In"  element="ghns:chkRequest" />
<output messageLabel="Out" element="ghns:chkResponse" />
<outfault ref="tns:InvalErr" messageLabel="Out"/>
</operation>
```

Fig. 3. The Excerpt WSDL of simple loan service.

```
val chkRepesponse : int = 0;  -- default 0 is invalid value of return
fun opCheckLoanCredit (loanType,requestAmt) =
     if loanType<> null andalso  loanType<> null  then
                    val SetLoanType = [ "EML", "NML"]
    if loanType  in SetLoanType then
     if requestAmt <= 999 then
             chkRepesponse = 0; --invalid
       else if (requestAmt >= 1000) andalso  (requestAmt <= 100000)
              then
               …… } –fill in the detail of valid case;
               chkRepesponse = xxxxx.00;
       else if requestAmt >= 100001 then
             chkRepesponse = 0; --invalid
   else if loanType not in SetLoanType then
           if requestAmt <= 999  then
             chkRepesponse = 0; --invalid
       else if (requestAmt >= 1000) andalso  (requestAmt <= 100000)
              then
              chkRepesponse = 0; --invalid
        else if requestAmt >= 100001
              then
              chkRepesponse = 0;  --invalid
else
     chkRepesponse;
```

Fig. 4. The dummy web service using ML language.

The rough skeleton of dummy web service is represented by the number of possible cases derived from the combination of the number of equivalence classes of each parameter. The designers have to fill in the details in each

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

possible case. When the orchestration abstraction and the dummy web service are completed, for successfully model abstraction, both parts would be composted as one CPN model. Then CPN tool is used to verify the CPN model with specifying the desired properties.

V. Implementation

This section describes the detail of the process implemented using a case study. The composite service of the loan credit calculation process is new web service attempting to combine the existing web services. The first activity is reviewing the infallibility of requesting loan, requested documents those are submitted over the Coop-Net banking system. The system requires related documents and basic information such as loan type, requested amount, term of payment, and the photocopy of identification card and so on. In case of a requester omits a required document or submits an incorrect document, this loan request would be rejected. If the documents are accurate and complete, the next step would be the checking of share values and the deposit balance. Both mentioned steps can be executed simultaneously. When the executions are complete, the summation of their values as property is used for comparing with the loan requested amount. If it is less than or equal the requested amount, the system uses the term of payment and the insurance code for requesting the insurance web service for adding value to the property value. Then the system compares the property value with loan requested amount again. If the property is less than the requested amount, this loan request would be rejected. When the property value checking process is completely executed and the property value is valid, the results of loan request processing are sent for credit scoring calculation. The results of this activity are a maximum credit, interest rate and installment type. After that, the loan request is sent to the fund committee for approval. The three out of five fund committee members should authorize the condition for the loan request. Then this loan request is inquired to create the new loan contract and the acknowledgment message is sent to the requester. The overview of new loan process is illustrated in Fig. 5

This loan process is written in BPEL process using eclipse BPEL designer. It is created by the begins with a process element defined with the name and target namespace, then all components of process are defined such as partner links, global variables, activities, flow and so on. This loan process has one partner link; it is the insurance web service that is provided by an outside service provider.

For model abstraction, we have developed simple application using xml parser for elements extraction. The application is used for extracting elements in the BPEL for creating the orchestration abstraction, which the elements are marked for mapping these to a place or a transition in the CPN model. WSDL file of insurance web service is used for dummy web service creation. The implementations of this dummy web service are programmed in ML language as a function. We include the dummy web service in CPN model, which the function of the dummy web service is called on the *outputArc*. For completed model, we describe the details in model such as colored sets, guard conditions, input arches, and output arches following the transformation rules, the CPN model is illustrated in Fig. 6.

We determine the sets of desired properties those are used for checking the CPN model such as a deadlock free, an unreachable path and an incorrect behavior. Model checking by CPN tool can simulate the execution and track each desired variable, which the faulty activities are represented with simulation mode or shown with red in model.

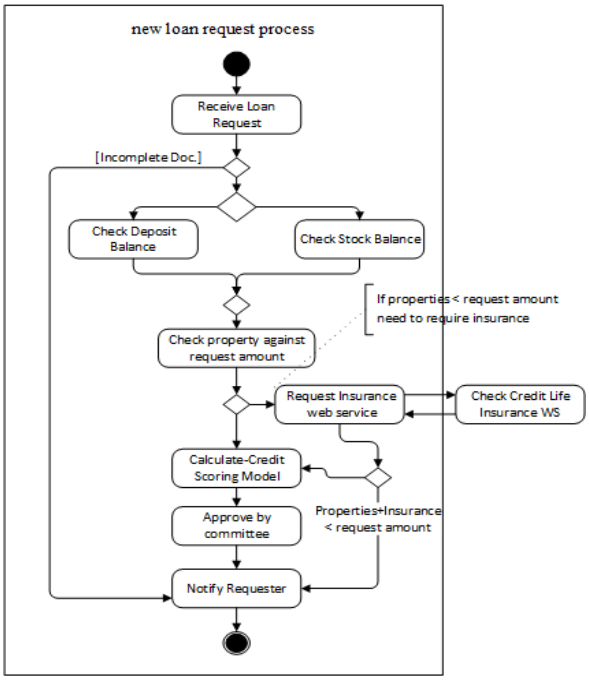


Fig. 5. Overview of loan request process.

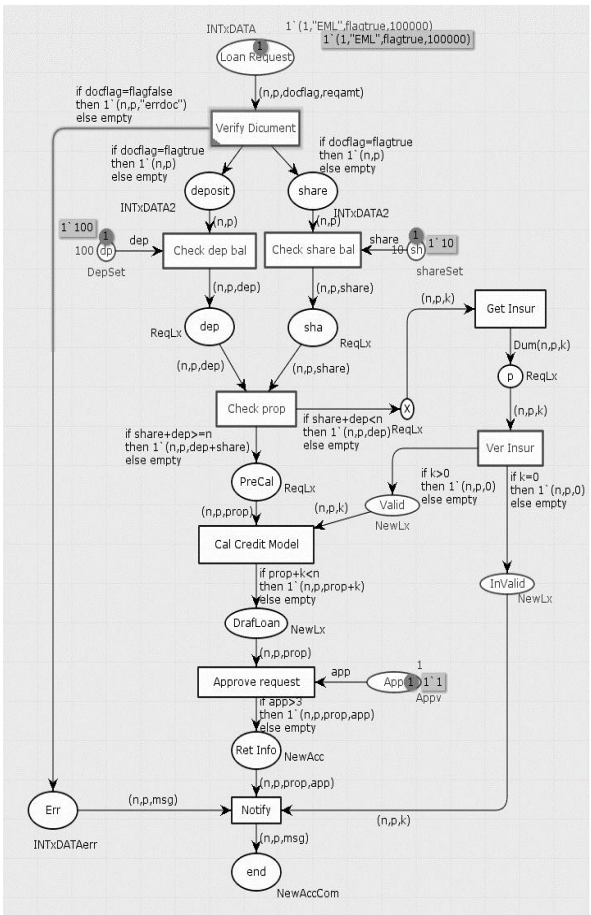


Fig. 6. Loan request process described as CPN model.

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

VI. Conclusion

Web service should be verified for achieving desired service properties. An example model correctness checking approach is the verifying with CPN tool. In composite web service, if some activity is the invoking or replying and also calling an outside service, it means that the designer maybe unable to catch its original implementation. We propose the method for verifying composite web service written in BPEL process in the early stage of the high level design process using the formal verification of the BPEL and its generated stubs, without its associated individual web services. This method produces two parts those are the orchestration and the dummy web service. We present the simple rules of the transforming of BPEL to CPN model; the equivalence class partitioning technique is particularly focused for stub generation which uses the WSDL of individual service. They are correctly and consistently transformed and verified in CPN Tool. This method is an expressive inscription for the places and the transitions expression in CPN model. The dummy web service not only is a part of verification but also acts as the original service that supports the designer to fill in the details including the controller of return value. The limitation of the dummy web service is its output only representing valid or invalid class. Our future work will include applying the proposed method to many service partners and many input parameters and attempt to create an automated system.

References

[1] M. N. Huhns and M. P. Singh, “Service-oriented computing: key concepts and principles,” IEEE Internet Computing, vol. 9, pp. 75-81, Jan-Feb 2005.

[2] OASIS, “Web Services Context Specification (WS-Context) Version 1.0,” 2015. [Online]. Available: http://docs.oasis-open.org/ws-caf/ws-context/v1.0/wsctx.pdf.

[3] OASIS, “Web Services Business Process Execution Language Version 2.0,” 2015. [Online]. Available: http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

[4] E. Newcomer, “Understanding Web Services-XML, WSDL, SOAP and UDDI,” Pearson Education, 2002.

[5] C. Kankanamge, "Web Services Testing with soapUI," Packt Publishing, 2012.

[6] V. Gehlot and C. Nigro, “An introduction to systems modeling and simulation with Colored Petri Nets,” proceedings of Simulation Conference (WSC), pp. 104–108, 2010.

[7] K. Jensen, L. M. Kristensen and L.Wells, “Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems,” Springer Published online: 13 March 2007, pp. 213–254, 2007.

[8] L. M. Kristensen, "State Space Methods for Coloured Petri Nets," Department of Computer Science, University of Aarhus, Aarhus, Denmark, Ph.D. Dissertation 2000. CPN Tools. [Online]. Available: http://cpntools.org/.

[9] M. Perepletchikov “A Formal Model of Service-Oriented Design Structure”, Software Engineering Conference, 2007. ASWEC 2007. 18th Australian, pp.71 -80.

[10] H. Guan, S. Ying and C. Wang, “A Correctness Verification Approach of the BPEL Exception Handling CPN Model Based on Temporal Property,” Journal of Networks, Vol.9, pp. 2743-2750, 2014.

[11] 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,” IEEE Trans. on Automation Science and Engineering, Vol.6, pp. 94–106, 2009.

[12] Y. Yan, P. Dague, Y. Pencole and M. -O. Cordier, “A Model-based Approach for Diagnosing Faults in Web Service Processes”, JWSR. , vol. 6, pp. 87-110, 2009.

[13] S. Hallé, “Automated Generation of Web Service Stubs Using LTL Satisfiability Solving,” WS-FM 2010, pp. 42-55.

ISBN: 978-988-19253-8-1
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)