ON V&V OF WEB SERVICE ORIENTED ARCHITECTURES

Abdelkader Dekdouk & Tarek H. El-Basuny

College of Computer Science and Engineering,
King Fahd University of Petroleum and Minerals,
KFUPM # 37, Dhahran 31261, Saudi Arabia, Mail Box 413,
Tel: 966-3-860-1967 & Fax: 966-3-860-2174,
Emails: dekdouk@kfupm.edu.sa , helmy@kfupm.edu.sa

Abstract

Now days it is widely accepted that the production of high quality software system requires a combination of techniques of testing (validation) and formal verification. These techniques have been widely applied in the development process of different kinds of software architectures. Recently service oriented architectures (SOA) have been emerged as a promising paradigm for supporting distributed computing. Different research contributions adopting the technique of testing have been carried out on SOA. However few works using formal verification have been done on SOA. In this paper we present our approach of formal verification to deal with this problem and situate it w.r.t. other contributions found in the literature treating this specific area. Our approach uses the technique of model-checking and simulation to verify the Web services interactions. For that, we use the modeling language BPEL4WS to describe the orchestration of the execution of Web services. Then we convert it to Promela, an input specification language of the Spin verifier tool in order to execute a series of verification and simulation procedures. All this is illustrated with a case study.

Keywords: *Web service oriented architecture, Business processes, Validation, Model-based Verification.*

1. INTRODUCTION

In the modern distributed application environment, component-based architectures such as COM and CORBA [14, 15] have been the best way to create robust, maintainable and scalable systems for the past ten years. However these models suffered from a few difficult problems. Different programming languages are not compatible as one might like. For instance a component written in C++ for use in a C++ environment sometimes has difficulty being used in an environment where Visual Basic is the main language. But even if the cross-language problems are fixed, still remain a bigger problem. It is difficult to overcome the problem of crossing heterogeneous platforms. That is calling a COM object from a Java program, or a CORBA object from a Visual Basic application. Cross-platform interoperability is not easy with the component models we have been using for the past decade. Other problems arise when calling a foreign object from beyond the firewall. Recently service oriented architectures have been emerged for supporting distributed computing and to remedy to the problems faced by component-based architectures. SOA are based on the concept of Web services that are service interfaces described with an XML meta-language like WSDL [17] or DAML-S [18]. Those Web services interact using XML messaging protocol like SOAP [19]. A lot of works have been focused on how to build such architectures [20, 21, 22, 23], and less work has been invested how to verify and validate such systems could be done [1, 2, 29, 30]. Even in this research domain we can claim that the works on formal verification [8, 9, 10] is less comparatively to what has been tackled in the side of validation [3, 4, 5, 6, 7]. The purpose of this paper is to present our approach of formal verification that has been initiated in [10]. This approach permits to verify the composition of Web services represented as business processes and the interactions between these business processes. For that we define an automatic conversion relationship between the language BPEL [11] used to describe business processes, and Promela language [12, 13] widely used to describe processes of distributed systems. Therefore any BPEL description can be automatically translated to a Promela description which in turn is feed to XSpin [13] verifier tool in order to execute upon it different verification procedures. This paper is organized as follows: In Section 2, we present the related works in the domain of testing of SOA, followed in Section 3 by the works undertaken on formal verification of such systems. In Section 4, we present our approach of verification of SOA, particularly model-based verification of interacting BPEL Web services using the XSpin verifier tool. In Section 5, we outline some concluding remarks and future works.

2. TESTING SERVICE ORIENTED ARCHITECTURES

In this section we discuss some current integration and testing techniques of Web services and service oriented applications. In [3], the authors proposed an approach investigating how automatic testing can be done on SOA systems. They defined an XML meta-language to describe test cases for services integration and they conceived a prototype tool called SITT (Service Integration Test Tool). SITT can automatically test and monitor whether the workflow between multiple service endpoints really behaves as expected, by analyzing the message flow. Each service endpoint is associated with a test agent, once the test case is executed, the test agent read the result stored in a standardized log file. Then this information is parsed and sent to the master agent. The master agent store the messages in a test database, which are then analyzed by the test daemon against predefined test behavior described in XML meta-language. Other testing approaches in service oriented computing consist in testing Web services as black boxes. In [4] a tool that is called Coyote is developed to support this approach and consists of two parts: the test master and test engine. The test master allows testers to specify test scenarios via WSDL description of a Web service. The test engine interacts with the Web service under testing, by invoking it with the parameters specified in the test script, and then validates the Web service response with the expected result and logs assessing results. This work has been extended in [5], by presenting a concept of how WSDL could be extended to capture four kinds of information description: input-output dependency, invocation sequence, hierarchical functional description and concurrent sequence.

3. VERIFICATION OF SERVICE ORIENTED ARCHITECTURE

In the area of verification the works have mainly focused on the analysis of interaction of composite Web services described in a Web services orchestration language [11, 18, 24]. In [27] the authors take a subset of the language DAML-S and provide it with an operational semantics described using Petri Nets. The tool KamaSim [25] is then used to carry out a series of simulations and property verifications on Petri Nets descriptions. The verification approach undertaken in [9] consists first in specifying a set of Web services requirement with a series of Message Sequence Charts. The composite Web services, is implemented with a BPEL process which is verified against the Web service requirement specification. The approach defined in [26] provides a subset of BPEL with an operational semantics using guarded automata. The latter formalism is then converted to Promela that is the input language for the Spin tool. This approach is similar to what we have introduced in [10], however in our work the conversion BPEL-Promela is done without passing by an intermediate formalism.

4. OUR APPROACH OF VERIFICATION

4.1 The language BPEL4WS

BPEL4WS [11, 28] is a specification language that models the business process behavior based on Web services. The business process can be of two kinds: executable processes and abstract processes or business protocols. Executable processes model actual behavior of a participant in a business interaction. Abstract processes specify the external behavior of the participant interacting with each other. Precisely they describe the mutually visible message exchange behavior of each of the participant involved in the protocol, without revealing their internal behavior. The BPEL business process XML-syntax along with short element definitions (provided as comments) is given as follows:

```
< process ... >
<partners>...< /partners>
<!—services, the process interact with -->
<containers>...</containers>
<!-- Data used by the process -->
<correlationSets>...</correlationSets>
<!-- Used to support asynchronous interactions -->
<faultHandlers>...</faultHandlers>
<!-- Alternate execution path to deal with faulty conditions -->
<compensationHandlers>...</compensationHandlers><!--Code to execute when "undoing" action-->
(activities)*
<!--The internal behavior of the process-->
</process ...>
```

The activities can be inductively defined as a set of basic activities that can be combined with control flow elements. Some of the basic activities are: <*invoke*> that permit to a process to invoke an operation on a partner. <*receive*> allows a process to receive an invocation from a partner. <*reply*> permits to a process to send a reply message corresponding to a partner invocation. <*assign*> performs the data assignment between containers. The control flow elements are: <*sequence*> that permits to execute activities sequentially. <*flow*> allows the execution in parallel of a set of activities. <*while*> allows a set of activities to execute iteratively while a certain condition is satisfied. <*pick*> permits to execute one of the several activities that are guarded by events. <*link*> defines a synchronization dependency between a source activity and a target activity. <*switch*> enables branching one activity from a set of conditional alternatives. For more details on the language we refer the reader to [11]. In the conversion BPEL-Promela, we focus more on the activities definitions that describe the actual behavior of the business process. Next, we present a case study showing that.

4.2 Conversion BPEL-Promela through a Case Study

This is an example of a loan approval business process combined of several Web services. First, customer information is received and his/her account is examined. If the amount of money in the customer's account is less than 10,000 then this loan request is transferred to a loan assessment Web service where the risk is assessed. Otherwise the loan request is approved and replied back to the customer. The assessment procedure checks, if the risk is '*low*' then the request is approved. Otherwise the loan request will not be accepted. The flow chart of this process is depicted in Figure 1.

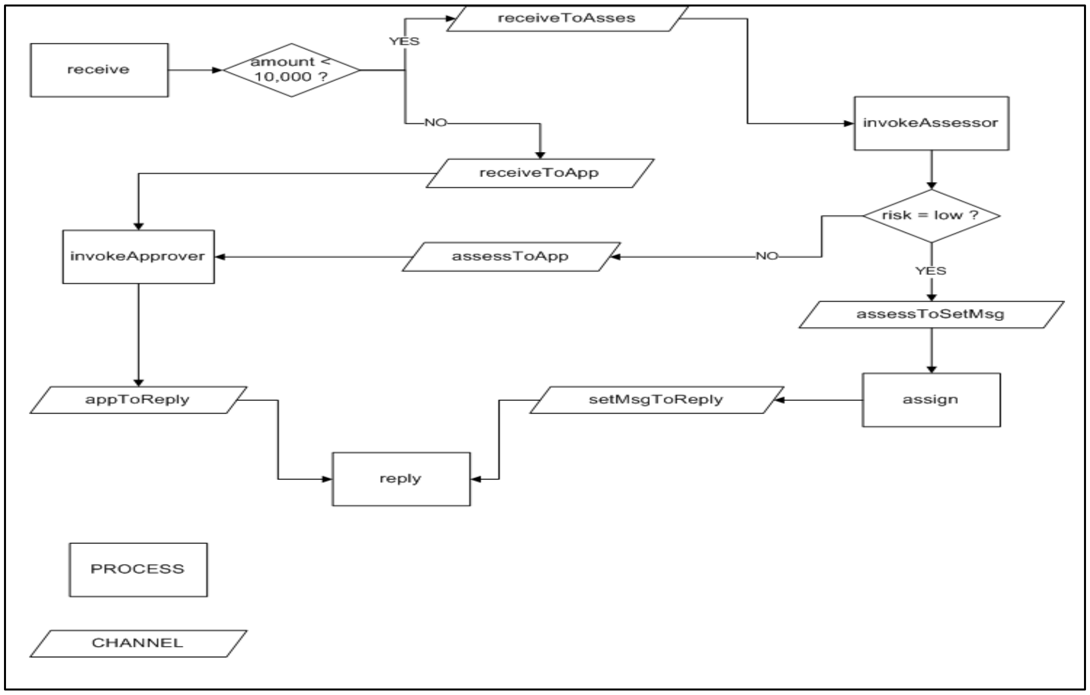


Figure 1: The flow chart of the loan approval process

The following (see Figure 2) is the BPEL code of the loan approval process. As shown by this description, the process defines four XML-data variable: *request*, *riskAssessment*, *approvalInfo* and *error* and it is interacting with three partners: *customer*, *approver* and *assessor*. The process represents a parallel execution of the following activities: *receive1* that is the task receiving an invocation of the operation *approve* with the data stored in the variable *request* from *customer*. If amount is less than 10000 then a link (a synchronization relationship) called *receiveToAssess* is generated otherwise a link *receiveToApp* is set and both channels have the task *receive1* as a source. The activity *invokeAssessor* representing the target of the link *receiveToAssess*, invokes the operation *check* with the input parameter *request*. The result of the operation must be delivered in the variable *riskAssessment*. If the content of *riskAssessment* is '*low*' then a response '*yes*' is assigned to the content of the element *accept* in the container *approvalInfo*. This is followed by the execution of the *reply* activity to the customer corresponding to the operation *approve* with the result stored in the variable *approvalInfo*. If the risk is not '*low*' or the account amount is greater than 10000 then *invokeApprover* invokes the operation *approve* from the partner *approver*, generating thus the data that is stored in *approvalInfo* and which is then replied to *customer*.

```
<?xml version="1.0" encoding="utf-8" ?>
<process name= "loanApprovalProcess"
            targetNamespace ="http://acme.com/loanprocessing"
            suppressJoinFailure = "yes"
            xmlns ="http://schemas.xmlsoap.org/ws/2002/07/business-process/ "
            xmlns:lns="http://loans.org/wsdl/loan-approval"
            xmlns:loandef="http://tempuri.org/services/loandefinitions"
            xmlns:asns="http://tempuri.org/services/loanassessor"
            xmlns:apns="http://tempuri.org/services/loanapprover">
<containers>
  <container name= "request"
messageType ="loandef:creditInformationMessage"/>
  <container name= "riskAssessment"
messageType ="asns:riskAssessmentMessage" />
  <container name= "approvalInfo"
     messageType="apns:approvalMessage" />
  <container name= "error"
messageType ="loandef:loanRequestErrorMessage"/>
</containers >
<partners>
  <partner name="customer"
    serviceLinkType ="lns:loanApprovalLinkType"
    myRole ="approver " />
  <partner name="approver"
    serviceLinkType ="lns:loanApprovalLinkType"
    partnerRole="approver" />
  <partner name="assessor"
serviceLinkType= "lns:riskAssessmentLinkType"
    partnerRole="assessor" />
</partners>
<faultHandlers>
```

```
  <catch faultName="lns:loanProcessFault"
    faultContainer="error">
  <reply partner="customer"
    portType="apns:loanApprovalPT"
    operation="approve" container="error"
    faultName="invalidRequest" />
  </catch>
</faultHandlers>
<flow>
<links>
     <link name="receiveToAssess" />
     <link name="receiveToApp" />
     <link name="appToReply" />
     <link name="assessToSetMsg" />
     <link name="setMsgToReply" />
     <link name="AssessToApp" />
</links>
<receive name="receive1" partner="customer"
  portType="apns:loanApprovalPT" operation="approve"
  container="request"
  createInstance="yes">
  <source linkName="receiveToAssess"
transitionCondition="bpws:getContainerData('request', 'amount')<10000"/>
  <source linkName="receiveToApp"
transitionCondition="bpws:getContainerData('request', 'amount')>=10000"/>
</receive>
<invoke name="invokeAssessor" partner="assessor"
  portType="asns:riskAssessmentPT" operation="check"   inputContainer="request"
  outputContainer="riskAssessment">
   <target linkName="receiveToAssess" />
   <source linkName="assessToSetMsg"
  transitionCondition="bpws:getContainerData(riskAssessment', 'risk') ='low'" />
  <source linkName="assessToApp"
  transitionCondition="bpws:getContainerData('riskAssessment', 'risk')!='low'" />
</invoke>
<assign name="assign">
  <target linkName="assessToSetMsg" />
  <source linkName="setMsgToReply" />
  <copy>
  <from expression="'yes'" />
  <to container="approvalInfo" part="accept" />
  </copy>
</assign>
<invoke name="invokeapprover" partner="approver"
  portType="apns:loanApprovalPT" operation="approve"
  inputContainer="request"
  outputContainer="approvalInfo">
  <target linkName="receiveToApp" />
  <target linkName="assessToApp" />
  <source linkName="appToReply" />
</invoke>
<reply name="reply" partner="customer"
  portType="apns:loanApprovalPT"
  operation="approve" container="approvalInfo">
  <target linkName="setMsgToReply" />
  <target linkName="appToReply" />
</reply>
</flow>
</process>
```

Figure 2: The BPEL description of the loan approval process

The strategy, we followed in the conversion BPEL-Promela is more focused on the description of the messages flow. It consists in mapping each basic activity to a process. Links represent synchronization relationships and hence they correspond to channels. In principle the containers correspond to variables of type XML-message. However due to the uniformity of this type and the expressivity restrictions we have in Promela types, we consider the containers as constants. The activities are controlled with the construct *<flow>* meaning that the processes run concurrently. In the case where the activities are executed sequentially (i.e. using the construct *<sequence>*), we create a channel of size 0 connecting the corresponding processes. For more details on the conversion BPEL-Promela, we refer the reader to [10]. Hereafter, we present the Promela description corresponding to the above BPEL loan approval process with a snapshot of the tool performing the automatic BPEL-Promela conversion and which has as front end XSpin verifier tool (see Figure 3).

```
mtype = {request, riskAssessment, approvalInfo, error, low};
mtype msg, risk;
byte amount=0;
chan receiveToAssess = [4] of { mtype };
chan receiveToApp = [4] of { mtype };
chan appToReply = [4] of {mtype};
chan assessToSetMsg =[4] of{mtype};
chan setMsgToReply = [4] of{mtype};
chan assessToApp =[4] of {mtype};
proctype receive1(){
msg = request;
do
            ::(amount < 10000) -> receiveToAssess ! msg;
            ::else-> receiveToApp ! msg;
od
}
proctype invokeAssessor() {
do
            ::receiveToAssess ? [msg]->receiveToAssess ? msg;
            if
            ::(risk = low) -> msg = riskAssessment; assessToSetMsg ! msg;
            ::else -> msg = riskAssessment; assessToApp ! msg;
            fi;
            ::else -> skip
od
}
proctype assign() {
do
        ::assessToSetMsg ? [msg] ->
             assessToSetMsg ? msg;
          msg = approvalInfo
         setMsgToReply! msg;
            ::else -> skip
od
}
proctype invokeApprover() {
do
            ::assessToApp ? [msg] -> assesToApp ? msg;
           msg = approvalInfo; appToReply ! msg;
            ::receiveToApp ? [msg]->receiveToApp ? msg;
           msg = approv alInfo; appToReply ! msg;
            ::else -> skip;
od
}
proctype reply() {
do
            ::appToReply ? [msg] ->appToReply ? msg;
       ::setMsgToReply? [msg] -> setMsgToReply ? msg;
            :: else -> skip
od
}
init {
run receive1();
run invokeAssessor_proc();
run assign();
run invokeApprover();
run reply();
}
```

4.3 Model Checking and Simulation with XSpin

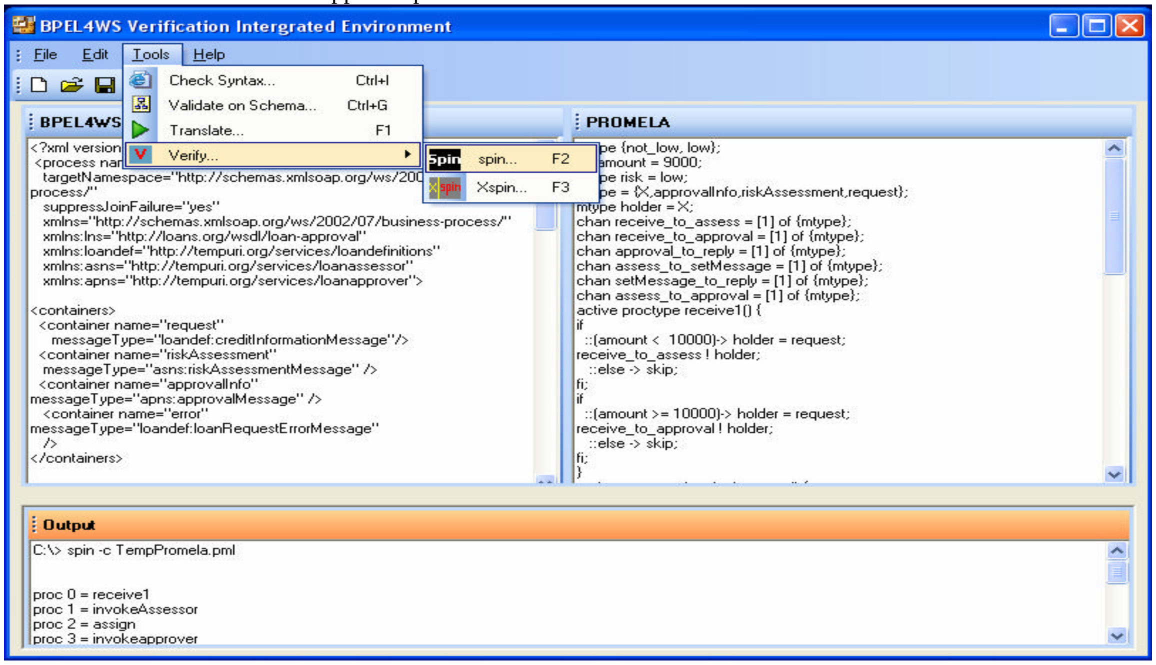
In this section we discuss the verification and simulation of the loan approval process. We take the Promela code corresponding to the loan approval business process; feed it to XSpin along with the specification of LTL properties to check them on our model. Consider the loan approval example, and the following basic properties: p = (msg = '*request*') and q = (msg = '*approvalInfo*'). Using these propositions, we can check for instance the liveness property [](p -> <> q) which means that always, if we have a message '*request*' sent through the channel *receiveToAssess* and contained in a variable named *msg* then eventually we receive a message '*approvalInfo*' through either the channel *setMsgToReply* or *appToReply* again contained in the variable *msg*. Checking this logical formula with XSpin on our Promela model using the integrated tool XSpin, ensures a liveness property of our BPEL model. In order to check the above liveness property, XSpin first negates the property then searches for any occurrence at some state where the property is true. If it's the case, then the property is satisfied. Hereafter is the result of the verification of the mentioned property on the loan approval model, which shows the satisfaction of the liveness property.

```
(Spin Version 4.1.3 -- 24 April 2004)
Warning: Search not completed
        + Partial Order Reduction
Full statespace search for:
        never claim             +
        assertion violations    + (if within scope of claim)
        acceptance   cycles     + (fairness disabled)
         invalid end states     - (disabled by never claim)
 State-vector 72 byte, depth reached 512, errors: 1(*the property   is found to be true*)
     277 states, stored (278 visited)
     250 states, matched
     528 transitions (= visited+matched)
       0 atomic steps
hash conflicts: 0 (resolved)
max size 2^18 states
```

Running the verification generated by XSpin with the option *-DSAFETY* to check for the cycles in the model, which eventually leads to *livelocks*, guarantees the safety of the model. Hereafter is the result of the safety verification, which shows that our model does not contain deadlocks behaviors.

```
(Spin Version 4.1.3 -- 24 April 2004)
        + Partial Order Reduction
Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        acceptance   cycles     - (not selected)
        invalid end states      +
State-vector 68 byte, depth reached 7013, errors: 0       (* means no deadlocks *)
   32460 states, stored
   89745 states, matched
  122205 transitions (= stored+matched)
       0 atomic steps
hash conflicts: 2352 (resolved)
(max size 2^18 states)
```

XSpin also allows us to simulate symbolically our models. In Figure 4, we present a snapshot of the simulation of our Promela model of the Loan Approval process.

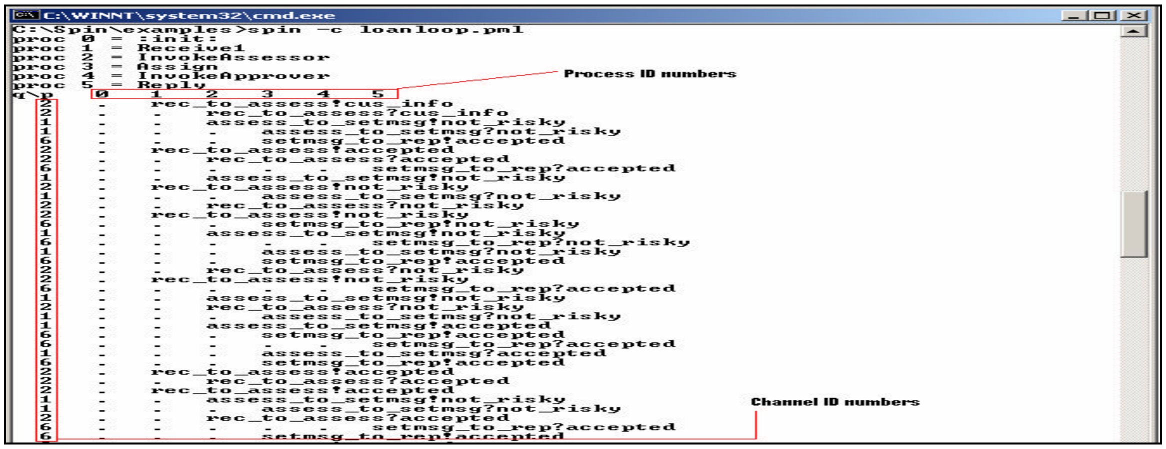


```
<?xml version
 <process nar
  targetNamespace="http://schemas.xmlsoap.org/ws/200
process/"
  suppressJoinFailure="yes"
  xmlns="http://schemas.xmlsoap.org/ws/2002/07/business-process/"
  xmlns:lns="http://loans.org/wsdl/loan-approval"
  xmlns:loandef="http://tempuri.org/services/loandefinitions"
  xmlns:asns="http://tempuri.org/services/loanassessor"
  xmlns:apns="http://tempuri.org/services/loanapprover">

<containers>
  <container name="request"
    messageType="loandef:creditInformationMessage"/>
  <container name="riskAssessment"
   messageType="asns:riskAssessmentMessage" />
  <container name="approvalInfo"
messageType="apns:approvalMessage" />
   <container name="error"
messageType="loandef:loanRequestErrorMessage"
  />
</containers>
```

```
pe {not_low, low};
amount = 9000;
pe risk = low;
pe = {X,approvalInfo,riskAssessment,request};
mtype holder = X;
chan receive_to_assess = [1] of {mtype};
chan receive_to_approval = [1] of {mtype};
chan approval_to_reply = [1] of {mtype};
chan assess_to_setMessage = [1] of {mtype};
chan setMessage_to_reply = [1] of {mtype};
chan assess_to_approval = [1] of {mtype};
active proctype receive1() {
if
  ::(amount <  10000)-> holder = request;
receive_to_assess ! holder;
   ::else -> skip;
fi;
if
  ::(amount >= 10000)-> holder = request;
receive_to_approval ! holder;
   ::else -> skip;
fi;
}
```

```
C:\> spin -c TempPromela.pml


proc 0 = receive1
proc 1 = invokeAssessor
proc 2 = assign
proc 3 = invokeapprover
```

Figure 3: A snapshot of the BPEL4WS verification integrated environment



```
C:\Spin\examples>spin -c loanloop.pml
proc  0 = :init:
proc  1 = Receive1
proc  2 = InvokeAssessor
proc  3 = Assign
proc  4 = InvokeApprover
proc  5 = Reply
q\p     0   1   2   3   4   5
  2     .   rec_to_assess!cus_info
  2     .   .   rec_to_assess?cus_info
  1     .   .   assess_to_setmsg!not_risky
  1     .   .   .   assess_to_setmsg?not_risky
  6     .   .   .   setmsg_to_rep!accepted
  2     .   rec_to_assess!accepted
  2     .   .   rec_to_assess?accepted
  6     .   .   .   .   .   setmsg_to_rep?accepted
  1     .   .   assess_to_setmsg!not_risky
  2     .   rec_to_assess!not_risky
  1     .   .   .   assess_to_setmsg?not_risky
  2     .   .   rec_to_assess?not_risky
  2     .   rec_to_assess!not_risky
  6     .   .   .   setmsg_to_rep!not_risky
  1     .   .   assess_to_setmsg!not_risky
  6     .   .   .   .   .   setmsg_to_rep?not_risky
  1     .   .   .   assess_to_setmsg?not_risky
  6     .   .   .   setmsg_to_rep!accepted
  2     .   .   rec_to_assess?not_risky
  2     .   rec_to_assess!not_risky
  6     .   .   .   .   .   setmsg_to_rep?accepted
  1     .   .   assess_to_setmsg!not_risky
  2     .   .   rec_to_assess?not_risky
  1     .   .   .   assess_to_setmsg?not_risky
  1     .   .   assess_to_setmsg!accepted
  6     .   .   .   setmsg_to_rep!accepted
  6     .   .   .   .   .   setmsg_to_rep?accepted
  1     .   .   .   assess_to_setmsg?accepted
  6     .   .   .   setmsg_to_rep!accepted
  2     .   rec_to_assess!accepted
  2     .   .   rec_to_assess?accepted
  2     .   rec_to_assess!accepted
  1     .   .   assess_to_setmsg!not_risky
  1     .   .   .   assess_to_setmsg?not_risky
  2     .   .   rec_to_assess?accepted
  6     .   .   .   .   .   setmsg_to_rep?accepted
  6     .   .   .   setmsg_to_rep!accepted
```

Figure 4: Simulation of Promela model of the loan approval example

5. CONCLUSIONS

In this paper, we have presented our approach of verification of BPEL model and situated it in the context of the different contributions undertaken in the area of verification and validation of Web services oriented architectures.

The concept of SOA is new and there exist many challenges for applying verification and testing techniques to SOA. Our next step is to finalize our prototype by extending our BPEL-Promela conversion prototype to some omitted BPEL constructs including the fault handler and the correlation elements. Another interesting direction for future research is to link the language BPEL to other verification and testing tools like TGV and TorX [31, 32] developed to deal with the distributed system testing.

6. ACKNOWLEDGMENTS

We would like to thank King Fahd University of Petroleum and Minerals for funding this research work and providing of the computing facilities, special thanks to anonymous reviewers for their valuable comments on this paper.

7. REFERENCES

1. Grumberg, O., Clarke, E. M., Peled, "Model Checking", MIT Press, Cambridge, MA, 1999.
2. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen, "Systems and Software Verification: Model-Checking Techniques and Tools", Springer Verlag, 2001.
3. S. Dustdar, S Haslinger, "Testing of Service Oriented Architectures–A practical approach", Net.ObjectDays, September 2004, Germany, Springer LNCS.
4. W.T.Tsai, R. Paul, W. Song, Z. Cao, "Coyote: An XML-Based Framework for Web Services Testing", 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02). Japan, Oct. 2002.
5. W.T. Tsai, R. Paul, Y. Wang, C. Fan, D. Wang, "Extending WSDL to Facilitate Web Services Testing", 7th IEEE International Symposium on High Assurance Systems Engineering, Japan, October 2002.
6. Y. Li, M. Li, J. Yu, "Web Services Testing, the Methodology, and the implementation of the Automation-Testing Tool", Grid and Cooperative Computing Workshop 2003, China, December 2003.
7. C. Fu, B. Ryder, A. Milanova, and D. Wannacott, "Testing of Java Web Services for Robustness", Proceedings of the International Symposium on Software Testing and Analysis, July 2004.
8. S. A. McIlraith, S. Narayanan, "Simulation, verification and automated composition of Web services", In Proceedings of the 11th International World Wide Web Conference. ACM, November 2002.
9. Foster, S. Uchitel, J. Kramer, and J. Magee. "Model-Based Verification of Web Services Compositions", Presented at Automated Software Engineering (ASE) Conference 2003, Canada, Oct. 2003.
10. Al-Gahtani, B. Al-Muhaisen, A-E-K. Dekdouk, "A Methodology and a Tool for Model-based Verification and Simulation of Web Services Compositions", IEEE Int. Conference on Informat ion & Computer Science, KSA, 2004.
11. Business Process Execution Language for Web Services (BPEL4WS), Version 1.1.
12. G. J. Holzmann, "Design and Validation of Computer Protocols", published by Prentice Hall in Nov. 1990
13. G. J. Holzmann, "The SPIN Model Checker: Primer and Reference Manual", Addison Wesley, Massachusetts, 2003.
14. Microsoft Corp. DCOM Technical Overview.
15. M. Fowler, D. Rice, M. Foemmel, E. Hieatt, R. Mee and R. Stafford. "Patterns of Enterprise Application Architecture", Addison Wesley Professional, 2002.
16. S. T. Albin, "The Art of Software Architecture", Design Methods and Techniques, Wiley, 2003.
17. E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, "Web service description language 1.1", W3c, 2001.
18. H. Zeng, A. Ankolekar, M. Burstein, "DAML-S: Semantic mark-up for Web services", In Proceedings of the International Semantic Web Working Symposium (SWWS 2001), November 2001.
19. M. Gudgin, M. Hadley, N. Mendelsohn, J-J. Moreau, and H.F. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework", W3C Recommendation. Technical report, W3C, June 2003.
20. J. Bloomberg, "The SOA Implementation Framework. The Future of Service-Oriented Architecture Software", Technical Report April 2004.
21. G. Miller, ".NET vs. J2EE", Communication of the ACM, 46(6):64-67, June 2003.
22. D. Lowe, et al. BizTalk(TM) Server: The Complete Reference. November, 2001.
23. Collaxa, "WSOS 2.0: An Introduction". http://xml.coverpages.org/CollaxaWP-200209.pdf
24. S. Thatte, "XLANG - Web Services for Business Process Design. Microsoft Corporation", May 2001..
25. S. Narayanan, "Reasoning about Actions in Narrative Understanding", Proc. International Joint Conference on Artificial Intelligence (IJCAI"99), pp. 350-358, 1999.
26. X. Fu, T. Bultan and J. Su, "Analysis of Interacting BPEL Web Services", Int. WWW Conference, June 2004.
27. S. Narayanan, S. McIlraith (2002). Simulation, Verification and Automated Composition of Web Services, Eleventh International World Wide Web Conference (WWW2002), Honolulu, May 2002.
28. S. Weerawarana, C. Francisco, "Understanding BPEL4WS, Part 1", IBM developer Works, August 2002.
29. S. Burton: Testing Safety-Related Software: a Practical Handbook, by S. Gardiner (Editor), Springer-Verlag, 1999 (Book Review). SW Testing, Verification Reliability 9(2): 135-136 (1999).
30. S. Burton, J. Clark, A. Galloway and J. McDermid, "Automated V&V for High Integrity Systems: A Targeted Formal Methods Approach", In the Proceedings of the 5th NASA Langley Formal Methods Workshop. June 2000.
31. Jard and T. Jéron, "TGV: theory, principles and algorithms", In Proc. of the 6th world conference on integrated design and process technology, 2002.
32. Automated Model Based Testing, J. Tretmans and E. Brinksma Editors: M. Schweizer, Progress 2002 - 3rd Workshop on Embedded Systems, STW Technology Foundation, Utrecht, the Netherlands, pp. 246 - 255, 2002.