ON V&V OF WEB SERVICE ORIENTED ARCHITECTURES

Abdelkader Dekdouk & Tarek H. ElBasuny

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 interactions 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
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.
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
<?xml version="1.0" encoding="utf-8" ?>
<process name="loanApprovalProcess" targetNamespace="http://acme.com/loanprocessing"
  suppressJoinFailure="yes"
  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>
</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).
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: \( p = (\text{msg} = \text{'request'}) \) and \( q = (\text{msg} = \text{'approvalInfo'}) \). Using these propositions, we can check for instance the liveness property \([p \rightarrow <> q]\) which means that always, if we have a message \('\text{request}'\) sent through the channel \(\text{receiveToAssess}\) and contained in a variable named \(\text{msg}\) then eventually we receive a message \('\text{approvalInfo}'\) through either the channel \(\text{setMsgToReply}\) or \(\text{appToReply}\) again contained in the variable \(\text{msg}\). 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 state space 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.

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.

**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