Model Based Testing of Web Service Composition

Poonkavithai Kalamegam and G.Zayaraz

Department of Computer Science and Engineering, Pondicherry Engineering College, Pondicherry, India.

Abstract. Web service composition is most mature and effective way to realize the rapidly changing requirements of business in service-oriented solutions. Web service composition testing is complex as it involves testing the interaction between the participating services as well as the business logic of the composite service. Colored Petri Nets (CPNs) provide a framework to verify web service compositions in data driven and control-flow driven approaches. In this paper an algorithm to generate test suite from the the data driven CPN model is proposed. The attributes like test prioritization, test suite size and redundancy in test design are used to gauge the effectiveness of the generated test sequences. The air line reservation system is used as case study to illustrate the test suite generation process.

Keywords : CPN, MBT, Web Service Composition Testing, Test Case Generation

1 Introduction

Service Oriented Architectures (SOA) is a design pattern composed of loosely coupled, discoverable, reusable, inter-operable platform agnostic services that follow a well defined standard. Web Services is a set of distributed message oriented interacting components that implements SOA. Several web services are composed to realize the business requirements. To test any composition, all the web services should be validated in isolation and also the interdependency between the services. Web services are composed by choreography or orchestration. In the past, the studies used test models like extensions of Control Flow Graphs, Finite State Machines and Petri Nets. Extending the usage of Colored Petri Nets beyond the system design phase, particularly in test design phase proves to be very effective in terms of greater test coverage and reduced test effort. In this paper a test case framework for web service composition testing is designed. The paper is organized as follows. In next section, we present the existing work on framework. In section III, we introduce the proposed algorithm. We present results and discussions in section IV followed by conclusions in the last section.

© Elsevier, 2012
2 Exiting Work

Several frameworks have been proposed for web service composition testing. A unit testing framework [1] was created for testing the BPEL compositions. The Timed Extended Finite State Machines (TEFSM) is used widely due to available tool support. Based on Time Extended Finite State Machine Cavalli [2] developed a testing framework that can be used in all phases of Web services composition. An abstract workflow-based framework was proposed by Karam [3] to test web service composition. A framework named Coyote [4] was created to support both test execution and test suite management. There exist few approaches to derive test cases from the CPN model. A simple approach to generate test cases using the state space was proposed in [5].

An efficient approach for building conformance test suite using the PN-ioco relation was proposed in [6]. U. Farooq et al [7] proposed a method to convert the AD activities to a CPN model and apply Random Walk Algorithm to create test sequences. Watanabe et al proposed two techniques that can be applied for concurrent systems [8]. On one hand CPN models have been used in test case generation but not in web service composition testing. On the other hand there are various approaches to verify web composition using CPN model. In particular, usage of CPN has not been explored to create test sequences for web service compositions. We aim at bringing in CPN models into MBT techniques to create cost effective and efficient test sequences for validating any composition.

3 Proposed Algorithm

The case study and draft version of the algorithm is present in [9]. The business requirements are represented by business processes consisting abstract activities with input/output data. The unique pairs can be easily derived from the pre conditions and post conditions specified in the business requirements. Moreover the pairs would be a limited and finite set. Hence the time taken to create the unique input output pairs (UIO-Pairs) would be minimal. Table 1 gives the exhaustive set for the case study taken into consideration. The CPN model is generated as a service net using the WSDL in the service portfolio and the business requirements. The approach to build the service nets is presented in [10].

Well-designed test data helps identify critical flaws in the functionality. The test data for each step in the sequence can be generated from the WSDL documents that are used to create the CPN model. The valid input space of a web service is the subset of the input space satisfying the precondition of the web service. The number of test sequences is directly related to the UIO-Pairs and the number of branches. The test sequences can also be prioritized by prioritizing the UIO-Pairs. The test suite size can be reduced by generating sequences for the UIO-Pairs that are business critical. In table 1 the first row is the most critical scenario where the end-user of the system is satisfied by receiving the tickets for the proposed itinerary. The fourth row pertaining to reservation cancellation due to over time has lowest priority. Therefore the test sequence generated using that UIO-Pair is also low. Row 5 is a special case where the
input is observable but the output is a Place; a state of the system. Such UIO-Pairs can be merged with one of the other test sequences that take similar input. The special cases are prioritized after the normal UIO-Pairs to enable merging of those test steps into already available test sequences accordingly. In this case the row 5 will be merged with the top priority test sequence.

### Table 1. UIO-Pairs

<table>
<thead>
<tr>
<th>SNo</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Proposed Traveler, TA</td>
<td>Itinerary</td>
</tr>
<tr>
<td></td>
<td>Credit Card No</td>
<td>Success (Tickets Booked)</td>
</tr>
<tr>
<td></td>
<td>Contact No</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Proposed Traveler, TA</td>
<td>Itinerary</td>
</tr>
<tr>
<td></td>
<td>Cancel Itinerary</td>
<td>Failure (Cancel Itinerary notification)</td>
</tr>
<tr>
<td>3</td>
<td>Proposed Traveler, TA</td>
<td>Itinerary</td>
</tr>
<tr>
<td></td>
<td>Credit Card No</td>
<td>Failure (Cancel Reservation notification)</td>
</tr>
<tr>
<td></td>
<td>Cancel Reservation</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Proposed Itinerary</td>
<td>Failure</td>
</tr>
<tr>
<td></td>
<td>OverTime</td>
<td>(Timeout notification)</td>
</tr>
<tr>
<td>5</td>
<td>Proposed Itinerary</td>
<td>Received</td>
</tr>
<tr>
<td></td>
<td>TA Reserved</td>
<td>(TA changed details)</td>
</tr>
</tbody>
</table>

In the CPN model the decision point is represented by a place with multiple outgoing arcs. The path to the decision point is common to all the branching out paths. Hence saving the common path reduces effort and redundancy. For example the test sequence for row 3 of table 1; for cancel reservation of ticket the seat must have done reservation in the airline and travel agent services, the steps for which is available in test sequence generated for row 1 of table 1. The proposed algorithm will result in high coverage with minimal effort. The algorithm is analyzed using the coverage criteria stated in [10] and the below defined criteria.

**Implementation Coverage (IMCov)**

The CPN model is generated using the WSDLs of the web services that participate in the web service composition. The implement coverage is defined as the ratio of WSDLs covered by the test suite and total number of WSDLs participating in the implementation of the web service composition.

\[
IMCov = \frac{\text{No. of WSDLs covered by test suite}}{\text{Total No. WSDLs part of implementation}}
\]

The below function is the logic for saving the decision points and the common path till that decision. This logic used when a ‘Place’ with multiple arcs is parsed. This function reduces the redundancy in test sequences thereby saving the time of parsing the CPN model again and again from the same starting point.
Function Analyze_Place(Place P)
Begin
If (P has multiple arcs) Then
    /*Decision points are Place with multiple arcs */
    Save decision points; DP=DP+P
If (this is first decision point) Then
    Path_Start = Initial Transition
Else
    Path_Start = Previous DP
End If
Path_End = Current DP
Next_Trans= Transitions that can result from P
Save test steps to Place along with pre-conditions
End If
End

In the CPN model for web service composition, a test sequence is any path from one of the initial state to one of the final state. In the Airline Reservation case study, the ‘Change Itinerary’ flow takes up input as Proposed Itinerary but the output depends on the ‘Check Seat Availability’ operation. The test sequence for this UIO-Pair is merged with the existing test sequence. This approach reduces the test sequence count and also avoids test step redundancy. Moreover the traceability is created to business requirements and the web services. To best of our knowledge traceability has not be considered in existing approaches.

The CPN model used is a data driven model that bridges the gap between the service and business domain. The model has already validated the accuracy of the system’s functional design. Generating test cases for such a model is bound to produce effective and efficient test cases providing maximum test coverage of the system under test. Figure 1 gives the system architecture for the test sequence generation process of web service composition.

![Fig 1. System Architecture](image-url)
**Algorithm CTS-G**

Begin

Initialize the CPN Model

For each UIO-Pairs Do

Let IP= input set of the UIO-Pair

Let OP= output set of the UIO-Pair

/*Reduce redundant traversal to decision point*/

If (IP part of existing DP) Then

Copy test steps till DP

T = Transitions after DP

End If

Remove used inputs from IP

Enable T that satisfy one or more IP

While (P Not OP) Do

/*Places will specify the inputs need for firing a transition*/

Choose an enabled transition (T) that influence OP

Fire (T)

/*Create test step for the test sequence relevant to UIO-Pair*/

If (P belongs to IP) Then

Record the Places connected to T as input

Remove P from IP

End If

Record traversal as test sequence

Record resulting Places and Arc expressions as output

If (T is web service operation) Then

/*Automatically update traceability matrix (TM) */

Update TM

End If

Analyze_Place(P)

End Do

If (OP is unobservable) Then

Merge test steps to suitable existing sequence

Update TCov, DCov

Else

Calculate TCov, DCov

Update test sequence with coverage and post-conditions

End If

Remove UIO-Pair

Initialize CPN

Next For

End
4 Results and Discussions

Farooq et al[8] has proposed an approach similar to ours. The test suite length for the airline reservation system case study was analyzed. The test sequences generated by the proposed system provide the same measure of coverage with lesser test suite size. The TSG technique creates separate test sequences for non-observable outputs also. As the number of such outputs increase the test suite size increases. In the proposed system such test sequences are merged to already existing test sequences, thus decreasing the test suite size. In Fig 2 the comparison of the test suite size is presented. The case study has only one non-observable output; hence the test suite size differs by 1.

![Test Suite Size Comparison](image)

**Fig 2. Test Suite Size Comparison**

Table 2 presents an analysis by comparing the approaches that exist for automatic generation of test sequences from the CPN Model. Traceability refers to traceability of the test sequences. Usage refers to the domain or the applications that uses test sequences generated.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Traceability</th>
<th>Usage</th>
<th>Redundancy</th>
<th>Test Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIU et al [7]</td>
<td>To model</td>
<td>GUI Apps</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Farooq et al[8]</td>
<td>To AD and model</td>
<td>SOA</td>
<td>Limited</td>
<td>X</td>
</tr>
<tr>
<td>Watanabe et al[9]</td>
<td>To model</td>
<td>Concurrency</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Our Work</td>
<td>To Bus. Req and Web Services</td>
<td>SOA</td>
<td>Handled</td>
<td>Handled</td>
</tr>
</tbody>
</table>

**Table 2. Comparison - Approaches for Test Sequence generation from CPN**
5 Conclusion

Testing is the most critical and expensive phase of the software development life cycle. Generation of test sequences or cases is most challenging part of testing phase as an efficient test design can detect greater number of faults. Moreover around 40% of software testing cost is spent on test design. In this paper we have present an approach to reduce that cost by means of automating the generation of test sequences for web service composition. We first analysed the existing approaches to generate test cases from the CPN model. Then we analysed the usage of CPN in web service composition. Then finally we consider the data driven CPN model used for verifying the design of composition and UIO-Pairs created from the business process requirements as input to create test sequences. As opportunities for future work, on-the-fly test sequence prioritization and automatic tracing back to business requirements will be taken up. As an extension we would compare our approach described here with already existing approaches in terms of savings in effort, requirements coverage and fault detection effectiveness.

References