Asian Journal of Convergence in Technology Volume II, Issue II
Issn No.:2350-1146, I.F-2.71

Automatic Test Data Generation of Web Services through XML Schema

Abhishek Kumar
School of Computer Engineering
KIIT University, Bhubaneswar
Orissa, India
abhikumar695@gmail.com

Akhilesh Kumar Pandey
School of Computer Engineering
KIIT University, Bhubaneswar
Orissa, India
infoakhileshpandey@gmail.com

Farahnaz Rezaeian Zade
Fooland Institute Of Technology
FoolandShahr
Isfahan, Iran
fr.rezaeeian@gmail.com

***Abstract*— Service-oriented architecture (SOA) is an IT architectural style that supports service-orientation. Service-Orientation represents a way of thinking about business and IT. SOA removed the gap between software and business.The Web service is the common technology to implement SOA. This paper proposes an automatic test data generation approach of web services through XML schema. Our proposed approach ensures the traceability of each element presents in the XML schema by using XML tree table structure. Further, itidentifies the complexity of the nodes and use the propose algorithm to generate the test data.**

Keywords—SOA; Web Services; Test Data Generation;XML Schema

I. Introduction

Today,large enterprise is going through business transformation and they are adopting the service based IT model. SOA provides solutions for integrating diverse systems that support interoperability, loose coupling and reuse. In SOA based application services may invoke other services to full-fill client needs. Among these services, some services may evolve. This dynamic and adaptive nature of SOA makes major concerns about its reliability and fault-free implementation. So, testing is necessary for SOA based application. However, the SOA testing approach is completely different from traditional testing approach. In traditional testing approach test location is centralized and testing is mostly done by software provider. There is offline regression testing and static test case profiling. But, SOA testing approach supports collaborative testing. Here, verification is done among the service providers, service brokers and clients in a collaborative way. In SOA testing, testing location should be distributed, remote, multi-agent and multi-phase. SOA testing supports online regression testing where data collected dynamically. Reliability is ensured by dynamic profiling and group testing.

Manual test case generation is time consuming and these test cases lose their usability after some time because services are dynamically published, bound, invoked and integrated. Due to this feature quality of service (QoS) can be achieved only if test cases can be automatically generated and testing is executed, monitored and analyzed at run time.

The Web service is based on open standards such as HTTP and XML-based protocols including SOAP andWSDL, Webservices are hardware, programming language, and operatingsystem independent. Web services are powered by XML and three other core technologies: WSDL, SOAP, and UDDI. Before building a Web service, its developers create its definition in the form of a WSDL document that describes the service's location on the Web and the functionality the service provides. Information about the service may then be entered in a UDDI registry, which allows Web service consumers to search for and locate the services they need. Web service provides several technological and business benefits [3].

XML schema defines the data types and structures of web service documents. An XML schema is the information source for the generation of test data. Data type definition (DTD) as the schema language defining the XML document structure. The data type definition (DTD) document contains the rules and relationships between the elements. In XML schema data type could be classified into *primitive data type* and *derived data type.Primitive data types* contain *string, boolean, float, double, duration, time, date, gyearMonth, gYear, gMonthDay, gDay, gMonth,hexBinary, anyURI, QName*and *NOTATION* [6]. *Derived data types* contain *normalizedString, token, language,Name,ID,integer,nonPositiveInteger,nonNegativeIn-eger, long,int,short,byte,unsignInt, unsignShort* [2]. We can use various XSD constraints to customize our test data. *MaxLength, minLength, maxExclusive, minInclusive, whitespace, enumeration* etc. are some XSD constraints. For example, *maxLength*is used to specify the maximum number of characters. Similarly, *minLength* specifies the minimum number of characters. *maxInclusive* specify the upper bound where as *minInclusive* specify the lower bound. *Whitespace* can be used to handle spaces, tab and line feed. *Enumeration* defines the acceptable value list. We can customize our test data by applying various XSD constraints in XML instances.

II. Related Work

Offutt et al. [1] Present new approach to test web service based on data perturbation. Data perturbation is a process to modify request message, resending the modified request message and analyze the response message for correct behavior. Here, data perturbation is used to test peer-to-peer interaction between service. Two methods are introduced related to data perturbation: data value perturbation and interaction perturbation. This approach is applicable only between two components at a time. Tsai et al. [7] Compare traditional software testing and web service testing and propose web service group testing (WSGT) technique to test composite services. WSGT can be used at each level of web service testing to rank the unit web service or composite web service at that level. In WSGT, the voting service can automatically generate an oracle for each input according to majority principle. Bai et al. [4] Generate test cases of individual service automatically based on the WSDL. WSDL carries the data transmission information and interface operation information about a service. Here, test data generation and test operation generation are two perspectives to generate test cases. Test data are generated by analyzing the message data type according to XML schema syntax and test operation generation is based on operation dependency analysis. Tsai et al. [8] Propose several testability evolution criteria for SOA software. To evaluate the support to test SOA software these evaluation criteria serve as a reference for both service providers and application builders. Zhang et al. [5] Extends UML 2.0 activity diagram to describe the syntax and behaviors of BPEL. This paper map BPEL primitive activity to UML 2.0 activity diagram. Next, mapping done from BPEL structure activity to UML 2.0 activity diagram. This paper discusses the feasibility of testing using web service business process testing. Chen et al. [9] Present an approach to automatically generate executable test cases based on activity diagrams. Here, they use simple path coverage adequacy criteria. Simple path refers to a feasible execution in an activity diagram that start from the initial node and end with the final node. Before randomly generated test case is executed, its execution path in activity diagram is predicted by calculating data classifiers of decision nodes. Useless input that is unable to raise path coverage will be dropped. Boghdady et al. [10] Proposes an enhanced XML- based automated approach for generating test cases from activity diagrams. Activity dependency table (ADT) is created for each XML file which cover all the functionalities in the activity diagram but in a reduced form. Activity dependency table contains pre-condition and post-condition to carry the input, output and guard conditions of the removed nodes in the form of expression. Activity dependency graph is generated from activity dependency table (ADT) to show all the possible test paths. Test cases derived from behavior or instructional models are functional test case and they have the same level of abstraction as the model creating them. Full condition coverage criteria used with state chart or communication diagram. All basic path coverage criteria used with activity diagram.

III. PROPOSED TEST DATA GENERATION APPROACH

Test data generation is an important part to test the SOA based services. Quality of test data determines the efficiency of testing. Test cases are generated using this test data. Test data is the set of test input data and test output data. In this report, we propose a test data generation approach. Our proposed approach uses the XML schema file. The proposed test data generation approach follows the following steps:-

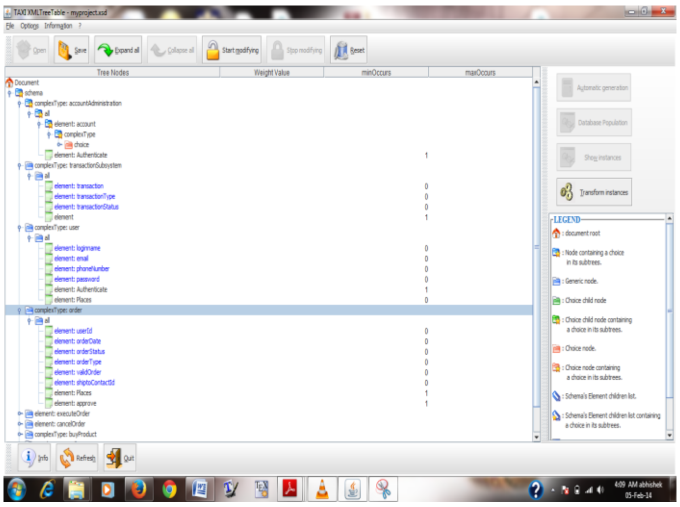
a. First, we get the XML document related to the service.
b. After getting the XML document we generate the tree table of the XML schema.
c. Further, we use our algorithm to generate the test data.

To implement our algorithm we use the example of 'Online Purchasing' an SOA based application. During the online purchasing we have to supply our credit card information. This credit card information is validated by the outside service vendor. Once the order has been finalized, the e-commerce company deliver our order and for this they have to co-ordinate with the shipping service vendor. So, 'Online Purchasing' provides a perfect understanding of SOA. We also use the TAXI tool [11] to generate the tree table of the XML schema. XML schema defines the data types and structures of web service documents. An XML schema is the information source for the generation of test data. Data type definition (DTD) as the schema language defining the XML document structure. The data type definition (DTD) document contains the rules and relationships between the elements.

Proposed Test Data Generation Algorithm

```
Input:dataTypeName, constraint.
Output: VALUE
Step 1. Generate tree table of the XML schema;
Step 2. Analyze the tree table at each node;
Step 3. If (data type is simple type){
Step 4. Set VALUE :=getData (dataTypeName, constraint);
} // data type is simple.
Step 5. else {
Step 6. Set VALUE :=getData (choice, constraint);
} // data type is complex.
Step 7. Write : VALUE;
Step 8. Exit.
```

To implement our algorithm we take an example of 'Online Purchasing' (Fig.5) an SOA based system. If needed, we can also divide the given schema into various sub schemas for reducing the complexity. Division of schema is made based on tag grouping. After getting the XML schema file of 'Online Purchasing' system we load this XML file into the TAXI tool and generate a tree table of the XML schema. Fig. 1 shows the tree table of the given XSD file. After getting the tree table of the XML schema we expand the tree of the XSD file and analyze the tree table at each node. If the node is a complex data type, then we convert this complex node into simple node. To make the complex node into simple node we convert the complex node elements into <choice> elements and apply various constraints on it. Applying constraints between choice elements help us to reduce the construction of a redundant combination of choices. There is a list of initial weights of children of a choice element. Default weight values for the element are given by TAXI automatically. The amount of the weights cannot be greater than 1 or smaller than 0. If it happens then TAXI sends an alert message and modification will be rejected. Under choice element sum of the weight must be within 1. If the user is not satisfied with the modified weight than he can click on the reset button. To start the modification user needs to click on start modification button. To lock the weight assignment user needs to click the stop modifying button. When the value of max occur is unbounded TAXI giving its default value 3. The default value can be changed.

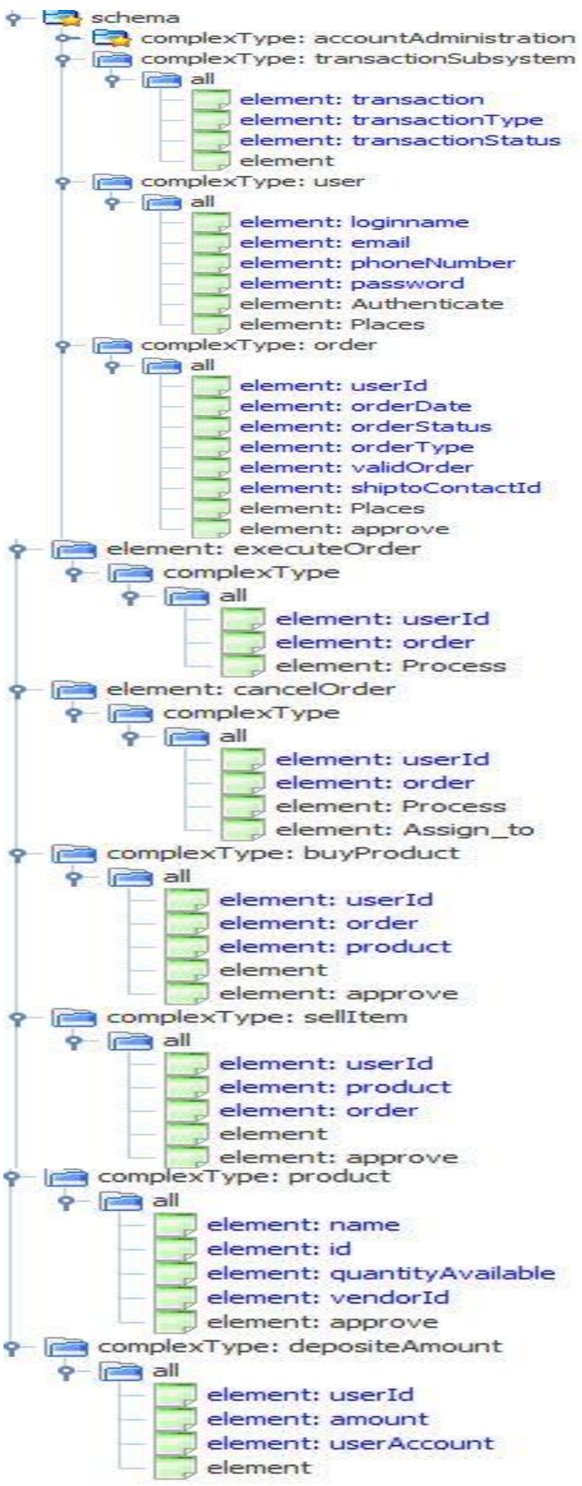


```
Tree Nodes                                   Weight Value   minOccurs   maxOccurs
Document
  schema
    complexType: accountAdministration
      all
        element: account
          complexType
            choice
        element: Authenticate                               1
    complexType: transactionSubsystem
      all
        element: transaction                                0
        element: transactionType                            0
        element: transactionStatus                          0
        element                                             1
    complexType: user
      all
        element: loginname                                  0
        element: email                                      0
        element: phoneNumber                                0
        element: password                                   0
        element: Authenticate                               1
        element: Places                                     0
    complexType: order
      all
        element: userId                                     0
        element: orderDate                                  0
        element: orderStatus                                0
        element: orderType                                  0
        element: validOrder                                 0
        element: shiptoContactId                            0
        element: Places                                     1
        element: approve                                    1
    element: executeOrder
    element: cancelOrder
    complexType: buyProduct
```

LEGEND
- : document root
- : Node containing a choice in its subtrees.
- : Generic node.
- : Choice child node
- : Choice child node containing a choice in its subtrees.
- : Choice node.
- : Choice node containing a choice in its subtrees.
- : Schema's Element children list.
- : Schema's Element children list containing a choice in its subtrees.

Fig. 1 Tree table of the 'Online Purchasing' system

The tree table presents in the fig. 1contains following nodes



```
schema
  complexType: accountAdministration
  complexType: transactionSubsystem
    all
      element: transaction
      element: transactionType
      element: transactionStatus
      element
  complexType: user
    all
      element: loginname
      element: email
      element: phoneNumber
      element: password
      element: Authenticate
      element: Places
  complexType: order
    all
      element: userId
      element: orderDate
      element: orderStatus
      element: orderType
      element: validOrder
      element: shiptoContactId
      element: Places
      element: approve
element: executeOrder
  complexType
    all
      element: userId
      element: order
      element: Process
element: cancelOrder
  complexType
    all
      element: userId
      element: order
      element: Process
      element: Assign_to
complexType: buyProduct
  all
    element: userId
    element: order
    element: product
    element
    element: approve
complexType: sellItem
  all
    element: userId
    element: product
    element: order
    element
    element: approve
complexType: product
  all
    element: name
    element: id
    element: quantityAvailable
    element: vendorId
    element: approve
complexType: depositeAmount
  all
    element: userId
    element: amount
    element: userAccount
    element
```

Fig. 2. Nodes of the 'Online Purchasing' system.

Each node presents in the *'Online Purchasing'* system represent an individual service. *transactionSubsystemservice,user service, order service, execute order service, cancel orderservice, by-product, service, sellItem service, product, service* and *depositeAmount* *service* are the different service components of *' Online Purchasing'* system.

User can enter the test data by double clicking on the element name. A window will appear where the user can see the existing values for that element. User can delete some existing values and add new ones one by one. Now we generate an XML instance for the subschema. An XML Instance document is a file that contains the information, or data, of what we are trying to describe with our schema. The instance document will refer to the schema design that provides its structure. When we are going to generate a test data we select the value for each element. Value can be generated randomly. We can also pick the value from the Test data set. Our proposed approach, focus on test data generation based on XML tree table structure. Here, we identify the complexity of the nodes and apply our algorithm to generate the test data.

We apply our test data generation method for *'product service'*.

```
<xsd:complexType name="product">
<xsd:all>
<xsd:element name="name" type="xsd:string"
 minOccurs="0" maxOccurs="1">
</xsd:element>
<xsd:element name="id" type="xsd:string"
minOccurs="0" maxOccurs="1">
</xsd:element>
<xsd:element name="quantityAvailable"
type="xsd:boolean" minOccurs="0" maxOccurs="1">
</xsd:element>
<xsd:element name="vendorId"
type="xsd:string" minOccurs="0" maxOccurs="1">
</xsd:element>
<xsd:element name="approve"
type="myproject:sellItem" minOccurs="1" maxOccurs="1">
</xsd:element>
</xsd:all>
</xsd:complexType>
```

Fig.3 XML schema format of *'product'* service component

Name, *id*, *quantityAvailable*, *vendorId*and *approve* are the different elements presents in the *'product service'*. We have to generate the test data for each element. Fig.3 shows the test data for product element*'name'*.

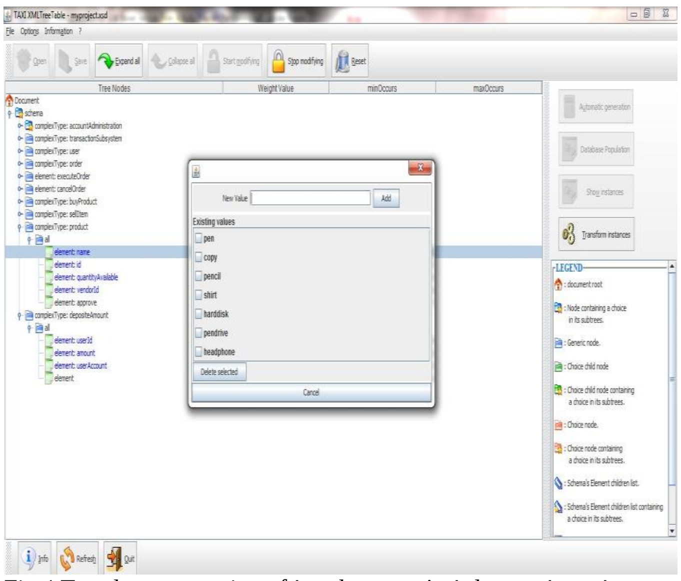


Fig.4 Test data, generation of *'product, service'* element *'name'*

We can bring the variety in the test data generation method by applying various XSD constraints.

We can apply <*maxlength*> XSD constraints to determine *maximum length* of the *'name'* element. This XSD constraint helps to generate a variety of test data as follows-

Length (name) >maxlength----------------------invalid test data.

Length (name) <maxlength----------------------valid test data.

Length (name) =maxlength-----------------------valid test data.

Length (name) =empty----------------------------invalid test data.

Similarly, we can generate the variety of test data for *'quantityAvailable'* element when the customer is going to purchase the product with some fixed quantity. For this we can apply <*minoccur*>XSD constraints.

When <*minoccur*=0> it shows invalid test data.

When <*minoccur*=1> it shows valid test data.

So, by applying various XSD constraints on element name of the service component we can generate the variety of the test data.

IV. Conclusion

This paper illustrates the automatic test data generation approach through XML schema. Our proposed approach uses the XML tree table structure. This tree table structure gives surety to traverse each element of the schema. Further, itidentifies the complexity of the nodes and use the propose algorithm to generate the test data. In our future work we will focus to generate the test case from this tree table structure with better test coverage.

References

[1] Jeff Offutt and Wuzhi Xu. Generating Test Cases for Web Services Using Data Perturbation. In IEEE (2003).

[2] Derived XML DataTypes, http://msdn.microsoft.com/en-us/library/ms256052.

[3] Erin Cavanaugh. Web services: Bene_ts, challenges, and a unique, visual development solution,Altova white paper, www.altova.com.

[4] Xiaoying Bai and Wenli Dong. WSDL-Based Automatic Test Case Generationfor Web Services Testing. In IEEE (2005).

[5] Zhang Guangquan, Rong Mei and Zhang Jun. A Business Process of WebServices Testing Method Based on UML 2.0 Activity Diagram. In IEEE (2007).

[6] Primitive XML DataTypes, http://msdn.microsoft.com/en-us/library/ms256220.

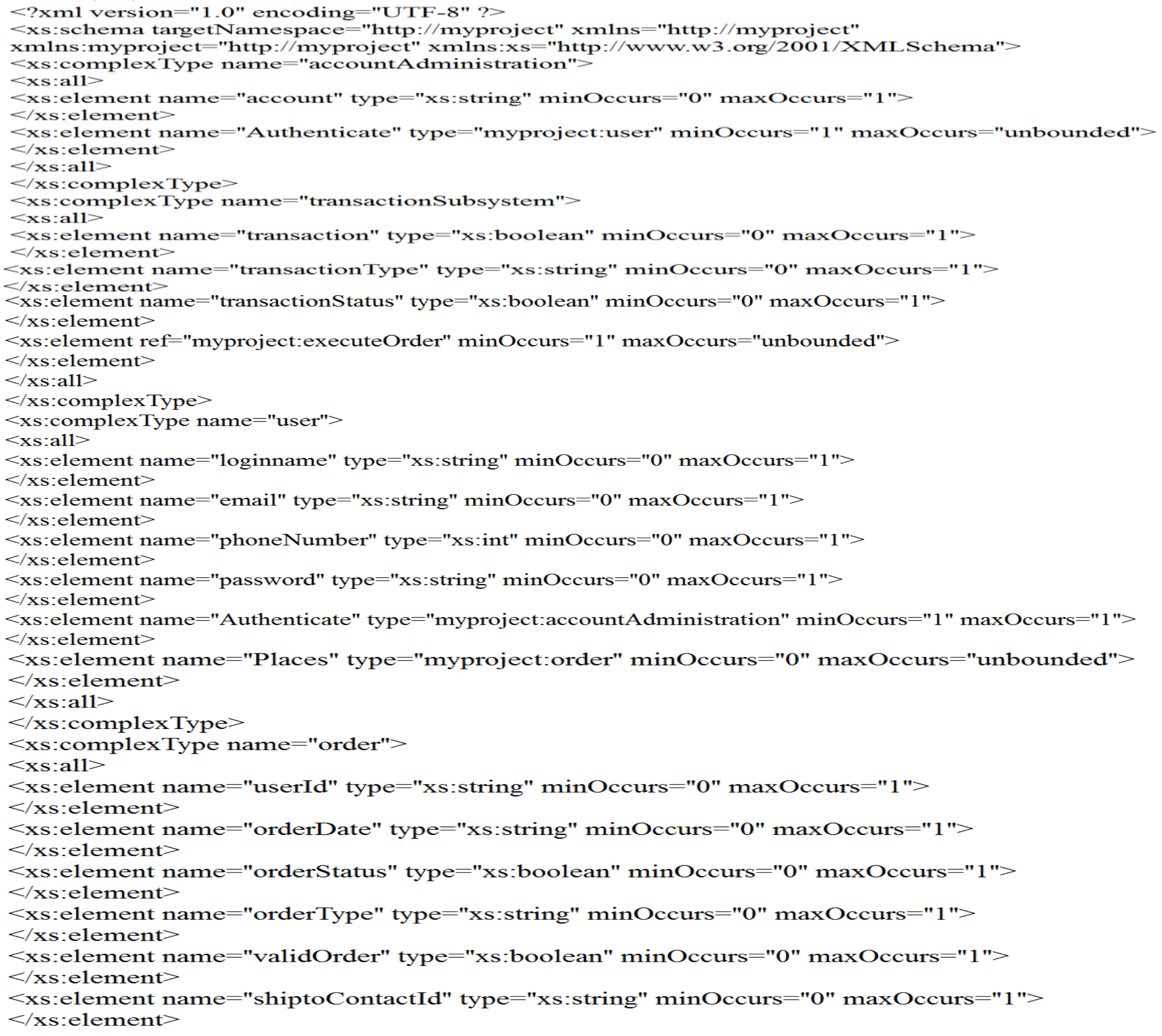
[7] W.T.Tsai, Y. Chen, R. Paul, N. Liao and H. Huang. Co-operative and Group Testing in Verification of Dynamic Composite Web Services. In IEEE (2004).

[8] W. T. Tsai, Jerry Gao, Xiao Wei and Yinong Chen. Testability of Softwarein Service-Oriented Architecture. In IEEE (2006).

[9] Xin Chen, Nan Ye, Peng Jiang, LeiBu and Xuandong Li. Feedback-Directed Test Case Generation Based on UML Activity Diagrams. In IEEE (2011).

[10] Pankinam N. Boghdady, Nagwa L. Badr, Mohamed A. Hashim and Mohamed F.Tolba. An Enhanced Test Case Generation Technique Based onActivityDiagrams. In IEEE (2011).

[11] Antonia Bertolino, Jinghua Gao, Eda Marchetti and Andrea Polini.Automatic Test Data Generation for XML Schema-Based Partition Testing. In proceedings of Second International Workshop on Automation of Software Test, IEEE , 2007.



```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema targetNamespace="http://myproject" xmlns="http://myproject"
xmlns:myproject="http://myproject" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:complexType name="accountAdministration">
<xs:all>
<xs:element name="account" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="Authenticate" type="myproject:user" minOccurs="1" maxOccurs="unbounded">
</xs:element>
</xs:all>
</xs:complexType>
<xs:complexType name="transactionSubsystem">
<xs:all>
<xs:element name="transaction" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="transactionType" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="transactionStatus" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element ref="myproject:executeOrder" minOccurs="1" maxOccurs="unbounded">
</xs:element>
</xs:all>
</xs:complexType>
<xs:complexType name="user">
<xs:all>
<xs:element name="loginname" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="email" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="phoneNumber" type="xs:int" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="password" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="Authenticate" type="myproject:accountAdministration" minOccurs="1" maxOccurs="1">
</xs:element>
<xs:element name="Places" type="myproject:order" minOccurs="0" maxOccurs="unbounded">
</xs:element>
</xs:all>
</xs:complexType>
<xs:complexType name="order">
<xs:all>
<xs:element name="userId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="orderDate" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="orderStatus" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="orderType" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="validOrder" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="shiptoContactId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
```

```
<xs:element name="Places" type="myproject:user" minOccurs="1" maxOccurs="1">
</xs:element>
<xs:element name="approve" type="myproject:buyProduct" minOccurs="1" maxOccurs="unbounded">
</xs:element>
</xs:all>
</xs:complexType>
<xs:element name="executeOrder">
<xs:complexType>
<xs:all>
<xs:element name="userId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="order" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="Process" type="myproject:transactionSubsystem" minOccurs="1" maxOccurs="1">
</xs:element>
</xs:all>
</xs:complexType>
</xs:element>
<xs:element name="cancelOrder">
<xs:complexType>
<xs:all>
<xs:element name="userId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="order" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="Process" type="myproject:transactionSubsystem" minOccurs="1" maxOccurs="1">
</xs:element>
<xs:element name="Assign_to" type="myproject:depositeAmount" minOccurs="0" maxOccurs="1">
</xs:element>
</xs:all>
</xs:complexType>
</xs:element>
<xs:complexType name="buyProduct">
<xs:all>
<xs:element name="userId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="order" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="product" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element ref="myproject:executeOrder" minOccurs="1" maxOccurs="1">
</xs:element>
<xs:element name="approve" type="myproject:order" minOccurs="1" maxOccurs="1">
</xs:element>
</xs:all>
</xs:complexType>
<xs:complexType name="sellItem">
<xs:all>
<xs:element name="userId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="product" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="order" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element ref="myproject:executeOrder" minOccurs="1" maxOccurs="1">
</xs:element>
<xs:element name="approve" type="myproject:product" minOccurs="1" maxOccurs="1">
</xs:element>
</xs:all>
</xs:complexType>
<xs:complexType name="product">
<xs:all>
<xs:element name="name" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="id" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="quantityAvailable" type="xs:boolean" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="vendorId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>|
<xs:element name="approve" type="myproject:sellItem" minOccurs="1" maxOccurs="1">
</xs:element>
```

```
</xs:all>
</xs:complexType>
<xs:complexType name="depositeAmount">
<xs:all>
<xs:element name="userId" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="amount" type="xs:int" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element name="userAccount" type="xs:string" minOccurs="0" maxOccurs="1">
</xs:element>
<xs:element ref="myproject:cancelOrder" minOccurs="0" maxOccurs="1">
</xs:element>
</xs:all>
</xs:complexType>
</xs:schema>
```

Fig.5 XML schema format of '*Online purchasing'* system