2008 International Conference on Computer Science and Software Engineering

WSDL-Based Automated Test Data Generation for Web Service

Chunyan Ma[1,2],Chenglie Du[2], Tao Zhang[1], Fei Hu, Xiaobin Cai
[1]College of Software and Microelectronics
[2]Institute of Computer Science
Northwestern Polytechnical University
127# Youyi West Rd. Xi'an shaanxi Province 710072, P. R. China
machunyan@nwpu.edu.cn

Abstract—**With the increase of the popularity of Web Service, more and more web applications are developed with this new kind of components. Web Service quality validation and control is becoming very critical to vendors, brokers and application builders. Both brokers and users usually have to use black-box testing because design and implementation details of Web Service are unavailable. This paper proposes a formal approach for automated test data generation for Web Service single operation based on WSDL specification. Firstly, we define a formal model for the datatype of input element of the operation. Secondly, the paper presents an algorithm that derives the model from WSDL. Thirdly, the paper presents a method that generates test data for operations from the model. Examples of using this approach are given in order to give evidence of its usefulness.**

Keywords- web service; automated test data generation;WSDL;

I. INTRODUCTION (HEADING 1)

Web Service-based software system may be composed of hundreds of these reused Web Services. However, if the quality of just a single of these Web Services is poor, it may greatly destroy the overall quality of the entire system. Web Service quality validation and control becomes very critical to vendors, brokers and application builders. Both brokers and users usually have to use black-box testing because WSDL specifications are available but design and implementation details of Web Services are not.

Testing Web Service includes two different levels for brokers and users, such as:

1) Testing the single operation of Web Service;
2) Testing the operations sequence of Web Service.

Regardless of stateless and stateful service [12], testing single operation of Web Service is necessary. This paper focuses on testing the single operation of Web Service. Compared with testing the single operation of other software components, testing Web Service is different because input element type of Web Service operation is based on XML schema datatypes and their associated constraining facets. Therefore this paper proposes an approach to automated test data generation for single operation of Web Service based only on the WSDL (Web Services Description Language) [1, 2]. This solution depends on input element type model which is a formal model of the XML Schema datatypes and their constraining facets [3, 4] that can be found in WSDL specification of the Web Service under test.

This paper is structured as follows. Section 2 presents a background on relevant concepts and other works in the area of testing Web Service. Section 3 presents a formal model and its automatic construction and proposes the test data generation algorithm based on the model, and also we present an application example. Remarks and outlook for future research are presented in Section4.

II. BACKGROUND AND RELATED WORKS

A. Background

WSDL is a XML-based language to define Web Services and the access to them. It specifies the location of the service and the operations or methods that the service exposes. The following is a pseudo-content model of WSDL description.

```
<description>
   <documentation />?
   [ <import /> | <include /> ]*
   <types />?
   [ <interface /> | <binding /> | <service /> ]*
</description>
```

The *Interface* is a set of operations. An operation is a sequence of input and output elements. Input elements are similar as the parameters of a function call in a traditional programming language.

The *types* define the datatypes of input element for each operation. Input element types could be defined in various schema languages. In this paper, we will only focus on the use of XML Schema datatypes since it's natively supported by WSDL2.0 for maximum platform neutrality. Based on the *types* definition, the input element type model is built in section 3.2.

XML Schema datatypes include built-in datatypes, simple datatypes and complex datatypes. Built-in datatypes are those which are defined in the XML Schema. Simple datatypes are those which are defined with the *<xs:simpleType…>* by user. A simple datatype could be obtained from a built-in datatype or another simple datatype by one of three means: by restriction, by list or by union. Complex datatypes are those which are defined with the <*xs:complextType*…> by user. Complex datatypes define all sub-elements with one of the key words *sequence*, *choice* or *all*. *Sequence* means that sub-elements may appear in the same order. *Choice* means that only one of sub-elements may appear. *All* means that sub-elements may appear in any order.

XML Schema Recommendation 2 identified five types of constraining facets, *unique*, *maxOccurs*, *minOccurs*, *nillable*, and *use*. They are constraining facets between complex datatypes and their sub-elements respectively. *MaxOccurs* and *minOccurs* are important for generating test data of input element, so the two types of constraining facets are considered in this paper. The constraining facets between simple datatypes and simple datatypes, and the constraining facets for built-in datatypes are classified as boundary constraints and non-boundary constraints which are important for generating test data of built-in datatypes. Table 1 lists them.

TABLE I. CLASSIFICATION OF THE CONSTRAINTS WHICH CONSTRAIN DATA VALUE

Classification of the constraints	Constraining Facets
Boundary Constrains	Length, maxLength, minLength, maxInclusive, maxExclusive, min-Inclusive, minExclusive, totalDigits, fractionDigits
Non-boundary Constraints	pattern, enumeration, whiteSpace

B. *Related Works*

A few papers [5, 6, 7, 8, 9] have presented testing techniques for Web Service single operation. Bai, et al. [5] and Jiang, et al. [6] also used WSDL for test data generation; nevertheless, their approach did not rely on the input element datatypes for single operation. The initial test data generation algorithm [6] can't handle complex and simple datatypes in XML Schema. Xu and Offutt [7, 8] used data perturbation to generate test data for Web Service. No automatic construction algorithm of XML Data Model from WSDL was presented. In the paper [9], test data generation is based on a formal abstract model of the XML Schema datatypes and their Constraining facets. However, the formal abstract model did not present the constraints between complex datatypes and their relative sub-elements and the constraints between simple datatypes and simple datatypes. These constraints are important for generating test data. In addition, no automatic construction algorithm of the abstract model from WSDL was presented. Our work is based largely on the previous work [7, 8, 9]. The contributions of this paper are:

- A formal type model for operation input element which presents XML Schema datatypes and various con-straining facets.
- An algorithm for input element type model construction from WSDL.
- An algorithm for test data generation from input element type model.

III. THE APPROACH

This paper proposes an automated test data generation approach for single operation of Web Service which is based on input element type model. The model aims to:

- Help in automatic test data generation.
- Provide a straightforward representation of XML Schema datatypes and various constraints.

The proposed steps for automated test data generation for Web Service operations are as follows:

Step 1: Preprocess the WSDL specification by parsing the *types* and the *interface* parts. For the *types* part, we remove *group*, *simpleContent* and *complexContent* and thus expand inline each occurrence of them, and then operation list for each interface is obtained from the preprocess. For each operation, we continue with Steps 2 through 4.

Step 2: Build input element type model for each input element of the operation.

Step 3: Obtain test data set for each input element based on the model which is built by Step 2.

Step 4: Generate test data set by input elements combinations for each operation.

Our main work lies in Step2 and Step3. Section 3.1 gives the definition of input element type model. Input element type model construction algorithm is presented in section 3.2. Section 3.3 describes the algorithm for test data generation for input element. An example of test data generation for the operation of order is presented.

A. *Input Element Type Model*

To automatically generate test data from WSDL, a formal model for XML Schema datatypes and their constraints is needed. Some researchers have presented formal models for XML. Xu, et al. [8] built an XML model to test Web Service by perturbing this model. Hanna Samer, et al. [9] proposed a formal abstract model of the XML Schema datatypes and their constraining facets. However, the model only presented the constraining facets for built-in datatypes. The model of this paper is different, because:

- Its definition and construction depends on the input element datatype of single operation in WSDL;
- It presents various constraining facets for XML Schema datatypes;
- It facilitates automated test data generation.

An input element type model of the operation can be modeled as a tree with constraints. We denote the model T=<N, D, n_r, IC, RC, EE, ED >, where:

- N is a finite set of complex datatypes child elements, child attributes or key word nodes in input element type definition in WSDL. The key word is the *sequence*, *choice* or *all*, which has an effect on test data generation.

- $D = S \cup B$. S is a finite set of simple datatypes nodes in input element type definition. B is the set of built-in datatype nodes in input element type definition.
- n_r is the root node. Input element will be modeled the root node.
- IC is a finite set of constraints between elements in $N \cup n_r$ and elements in N. $\forall x \in IC$, denoted as $x=<facet_1,value_1>$ or $x=<facet_1,value_1>$ && $<facet_2,value_2>$. The operator && denotes two facets "and" together. According to XML Schema Structures, $<facet_i, value_i>(1 \le i \le 2)$ is one of the following constraints: (1) <*maxOccurs*, *non-NegativeInteger*>; (2)<*maxOccurs*, *unbounded*>; (3)<*minOccurs*, *nonNegativeInteger*>. The default value for *minOccurs* and *maxOccurs* is "1". IC presents the constraints between complex datatypes and their respective sub-elements.
- RC is a finite set of constraints between elements in $N \cup n_r \cup D$ and elements in D. $\forall x \in RC$, denoted as $x=<facet_1,value_1>$ op_1 $<facet_2,value_2>$ op_2 … op_n $<facet_n,value_n>$, where op_i = && or op_i = ||, $1 \le i \le n$, The operator "||" denotes the facets pattern and enumeration "or" together. The operator "&&" denotes all other facets "and" together. Table 1 lists these facets. RC presents the constraints between simple datatypes and simple datatypes, and the constraining facets for built-in datatypes.
- EE is a finite set of edges. $\forall e \in EE$, denoted as $e(p,x,c)$, where $p \in N \cup n_r, c \in N, x \in IC \cup \{\phi\}$.
- ED is a finite set of edges. $\forall e \in ED$, denoted as $e(p,x,c)$, where $p \in N \cup D, c \in D, x \in RC \cup \{\phi\}$.

The edges in EE and ED make up two different kinds which correspond to two types of constraints in IC and RC. For an edge e(p, x, c) in EE and ED, $x = \phi$ is denoted as no constraint. The node 'c' is called the child of the node 'p'. The node 'p' is called the parent of the node 'c'. 'x' means that the node 'p' constrains the node 'c'.

B. Input Element Type Model Construction Algorithm

Figure 1 shows how to construct the input element type model from the *types* definition in preprocessed WSDL specification. In the generated model, Input element is modeled as root node of the model tree, and the nodes in B are the leaf nodes.

Figure 2 provides an example, the preprocessed WSDL specification for order, to implement the function of submitting the order. Figure 3 shows the type model tree of the input element *OrderRequest* generated by the algorithm presented in Figure 1.

C. Test Data Generation for input element

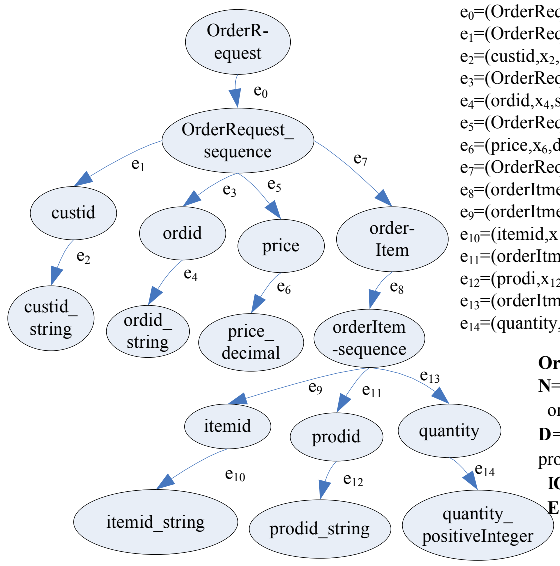
We first generate test data for built-in datatype nodes in the set B which are leaf nodes in input element type model tree,

```
Input: The type definition of input element 'x' in WSDL.
Output: The type model tree T of input element 'x'.
Step 1: (1) Let N=∅, D =∅, IC=∅, RC=∅, EE=∅ and ED =∅; (2) A new node 'n' is produced with the name of input element 'x'; (3) Let $n_r$ = n.
Step 2: (1) If the datatype of 'n' is a complex datatype then continue with Step 3; (2) If the datatype of 'n' is a built-in datatype then continue with Step 4; (3) Otherwise continue with Step 5.
Step 3: (1) A new node 'm' with one of n_ sequence, n_choice or n_all and an edge e(n, x, m) are produced, where x= ϕ; (2) Let N=N∪{m}, EE=EE∪{e}, n=m; (3) For each sub-element of the complex datatype, the new node 'm' with the name of the sub-element and an edge e(n, x, m) are produced, where 'x' is the facet which constrains the sub-element, and let N=N∪{m}, EE = EE ∪ {e}, IC=IC∪{x}, n=m, and then Continue with Step2.
Step 4: (1) We assume that the built-in datatype is b. A new node 'm' with n_b and an edge e(n,x,m) are produced, where x=ϕ; (2) Let B=B∪{m},ED= ED∪{e}.
Step 5:
IF 'n' is defined by using a built-in data type or another simple datatype as the restriction base THEN
  (1) A new node 'm' with the restriction base and an edge e(n,x,m) are produced, where 'x' is the facet for the base; (2) If the restriction base of 'n' is built-in datatype then let B=B∪{m}, ED=ED∪{e}, RC = RC ∪{x}; (3) If the restriction base of 'n' is another simple datatype then let S=S∪{m}, ED=ED∪{e}, RC = RC ∪ {x}, n =m, and continue with Step 5.
ENDIF
IF the simple data type of 'n' is a list data type THEN
  (1) A new node 'm' with the list item and an edge e(n,x,m) are produced, where 'x' is the restriction facet for the list; (2) Let S=S∪{m}, ED=ED∪{e}, RC=RC∪{x}, n =m; (3) Continue with Step 5.
ENDIF
IF the simple data type of 'n' is a union type THEN
  FOR each member in union type
      (1) Build a new node 'm' with member type, an edge e(n,x,m) is also produced, where x=ϕ; (2) Let B=B∪{m}, ED=ED∪{e}, RC=RC∪{x}, n=m; (3) Continue with Step 5.
  ENDFOR
ENDIF
```

Figure 1. Algorithm for input element type model construction

```
<wsdl:types>
<xs:element name="OrderRequest"
  <xs:complexType>
    <xs:sequence>
      <xs:element name="custid" minOccurs="0">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:maxLength value="10"/>
              <xs:minLength value="10"/>
          </xs:restriction>
        </xs:simpleType>
     </xs:element>
     <xs:element name="ordid">
       <xs:simpleType>
         <xs:restriction base="xs:string">
           <xs:maxLength value="10"/>
           <xs:minLength value="10"/>
         </xs:restriction>
       </xs:simpleType>
     </xs:element>
     <xs:element name="price" type="priceType"
                 minOccurs="0" />
      <xs:element name="orderItem" type="item"
                  maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:simpleType name="priceType">
  <xs:restriction base="xs:decimal">
    <xs:fractionDigits value="2">
    <xs:maxInclusive value ="10000"/>
  </xs:restriction>
</xs:simpleType>
<xs:complexType name="item">
  <xs:sequence>
    <xs:element name="itemid">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:maxLength value="10"/>
          <xs:minLength value="10"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="prodid" minOccurs="0"/>
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:maxLength value="10"/>
            <xs:minLengh value="10"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="quantity" minOccurs="0"/>
      <xs:simpleType>
        <xs:restriction base="xs:positiveInteger">
          <xs:maxInclusive  value="100"/>
            </xs:restriction>
            </xs:simpleType>
    </xs:element>
  </xs:sequence>
 </xs:complexType>
</wsdl:types>
<wsdl:interface name="Order">
  <wsdl:operation name="order" >
    <wsdl:input element="OrderRequest" />
  </wsdl:operation>
</wsdl:interface>
```

Figure 2. WSDL for order after preprocessing



e_0=(OrderRequest,x_0,OrderRequest_sequence)
e_1=(OrderRequest_sequence,x_1,custid)
e_2=(custid,x_2,string)
e_3=(OrderRequest_sequence,x_3,ordid)
e_4=(ordid,x_4,string)
e_5=(OrderRequest_sequence,x_5,price)
e_6=(price,x_6,decimal)
e_7=(OrderRequest_sequence,x_7,orderItem)
e_8=(orderItme,x_8,orderItme_sequence)
e_9=(orderItme_sequence,x_9,itemid)
e_{10}=(itemid,x_{10},string)
e_{11}=(orderItme_sequence,x_{11},prodid)
e_{12}=(prodi,x_{12},decimal)
e_{13}=(orderItme_sequence,x_{13},quantity)
e_{14}=(quantity,x_{14},positiveInteger)

x_0=ϕ
x_1=(minOccurs, "0")$\cup$ (maxOccurs,"1")
x_2=(maxLength,"10")$\cup$ (minLength,"10")
x_3=(minOccurs, "1")$\cup$ (maxOccurs,"1")
x_4=(maxLength,"10")$\cup$ (minLength,"10")
x_5=(minOccurs,"0")$\cup$ (maxOccurs,"1")
x_6=(fractionDigits,"2")$\cup$(maxInclusive,"10000")
x_7=(minOccurs,"1")$\cup$ (maxOccurs,"unbounded")
x_8=ϕ
x_9=(minOccurs, "1")$\cup$ (maxOccurs,"1")
x_{10}=(maxLength,"10")$\cup$ (minLength,"10")
x_{11}=(minOccurs, "0")$\cup$ (maxOccurs,"1")
x_{12}=(maxLength,"10")$\cup$ (minLength,"10")
x_{13}=(minOccurs, "0")$\cup$ (maxOccurs,"1")
x_{14}=(maxInclusive,"100")

OrderRequestT=(N，D，IC，RC，EE，ED，n_r)
N={OrderRequest, OrderRequest_sequence,custid, ordid, pirce, orderItem, orderItem_sequence, itemid, prodid, quantity}； n_r=OrderRequest
D=BUS； **B**={custid_string, ordid_string, price_decimal, itemid_string, prodid_decimal, quantity_positiveInteger}； **S**=Φ
IC={x_0, x_1, x_3, x_5, x_7, x_8, x_9, x_{11}, x_{13}}； **RC**={x_2, x_4, x_6, x_{10}, x_{12}, x_{14}}
EE={e_0, e_1, e_3, e_5, e_7, e_8, e_9, e_{11}, e_{13}}； **EC**={e_2, e_4, e_6, e_{10}, e_{12},e_{14}}

Figure 3. The type model tree of input element OrderRequest.

and then obtain test data for the nodes in the set N$\cup$S$\cup$$n_r$ from the bottom up according to the relationships among nodes in model. Finally, we can obtain the test data set of the node n_r for input element.

1) Obtain test data for built-in datatype nodes in the set B

In this section, we apply the constraints in RC to obtain test data for built-in datatype nodes. Figure 4 is the algorithm that obtains the test data for the built-in datatype nodes. If the parent of the node 'm' in B is not a simple datatype, we can directly get test data based on the facets which constrains the node 'm'. If the parent 'n' of the node 'm' in B is a simple datatype, we will iteratively decide whether the parent 'o' of the node 'n' is a simple datatype. The constraints between simple datatypes and simple datatypes will constrains the node 'm' in further constraining facets, so we must obtain test data based on the all constraining facets including the constraints between simple datatypes and simple datatypes for the node 'm'.

According to black-box testing strategy (e.g. boundary value, equivalence class, or random testing), we could build a knowledge base for the built-in datatypes and their constraints. In this knowledge base each XML Schema built-in datatype is associated with rules to generate test data based on the constraining facet and the value for that constraint [11].

```
Algorithm: obtaining test data set for each built-in datatype.
Input: T=<N, D, $n_r$, IC, RC, EE, ED>.
FOR each node 'm' in B and edge e(n,x,m)
  IF $n \in N$ THEN
     According to the black-box testing (e.g. boundary
     value, or equivalence class, or random testing), we
     generate valid or invalid test data for the built-in
     datatype 'm' with constraint 'x'.
  ELSE
     Let y=x;
     FOR e(o,x,n) and $o \in S$
        IF($x \neq \phi$) THEN
           y=y&&x;
        ENDIF
        n=o;
     ENDFOR
     IF e(o,x,n) and $o \in N$ and $x \neq \phi$ THEN
        y=y&&x;
     ENDIF
     According to the black-box testing, we generate valid
     or invalid test data for the built-in datatype 'm' with
     constraint 'y'.
  IFEND
ENDFOR
```
Figure 4. Deriving test data for the built-in datatype nodes in the model

TABLE II. TEST DATA FOR THE BUILT-IN DATATYPES

Built-in types	**Knowledge Base**	
	Constraining facet and constraint value	***Valid and invalid values***
(custid)string	(maxLength,”10”)&&(minLength, “10”)	xx00xx00xx, a001
(ordid)string	(maxLength,”10”)&&(minLength, “10”)	yy00371102, b001
(price)decimal	(fractionDigits,“2”)&&(maxInclusive,“10000”)	957.3, -978.56
(itemid)string	(maxLength,”10”)&&(minLength, “10”)	c001uddddd, c008888888e
(prodid)string	(maxLength,“10”)&&(minLength,“10”)	0371102345, 037110234
(quantity)positiveInteger	(maxInclusive,”100”)	45, 124

For the type model tree of input element *OrderRequest* in Figure 3, Table 2 lists the built-in datatypes, the constraints to which their values must conform, and the generated test data according to equivalence class testing. We divide the input domain into valid sub-domain and invalid sub-domain for each built-in datatype and select one from each sub-domain at random.

2) Relationships among nodes in model

In this section, we will apply the constraints in IC to fetch the relationships among nodes in the model. The relationships facilitate the process obtaining test data of input element from the bottom up with the model tree T. We assume that the model T=<N, D, n_r, IC, RC, EE, ED >. The relationships definition among nodes in input element type model is as follow:

- $\mathbf{m \rightarrow i_1 * n_1 + i_2 * n_2 + \ldots\ldots + i_j * n_j}$ **,** where $1 \leq i_j, j \leq 1$. For the node 'm', 'n_1' , 'n_2',….. , 'n_j' in T, the node 'n_1', 'n_2',….. , 'n_j' are the child elements of the node 'm'. The child node 'n_k' can occur i_k times (**$1 \leq k \leq j$**).
- $\mathbf{m \rightarrow sequence(i_1 * n_1 + i_2 * n_2 + \ldots\ldots + i_j * n_j)}$**,** where $1 \leq i_j$, $1 \leq j$. The child elements of the node 'm' may appear in a same order as in the formula.
- $\mathbf{m \rightarrow choice(i_1 * n_1 + i_2 * n_2 + \ldots\ldots + i_j * n_j)}$**,** where $1 \leq i_j$, $1 \leq j$. Only one of child elements of the node 'm' may appear.
- $\mathbf{m \rightarrow all(i_1 * n_1 + i_2 * n_2 + \ldots\ldots + i_j * n_j)}$**,** where $1 \leq i_j$, $1 \leq j$. The child elements of the node 'm' may appear in any order.

Figure 5 is the algorithm that obtains the relationships among nodes in the model T according to edges in EE and EC and the constraints in IC.

Combining all the possible values of *minOccurs* and *maxOccurs* defined for each child of complex datatype is infeasible. To settle this problem, here we associated a random value taken from the input domain with the occurrences of each element in the algorithm. We could also associate values taken from the input domain with the occurrences of each element according to boundary values, or equivalence. In the future we will evaluate which solution is better.

The relationships of input element OrderRequest in Figure 3 which are obtained by the algorithm presented in Figure 5 are as follow:

- orderRequest→orderRequest_Sequence
- orderRequest_Sequence→custid+ordid+price+ 3∗orderItem
- orderItem→orderItem_Sequence
- orderItem_Sequence→itmeid+prodid+quantity
- custid→custid_string
- ordid →ordid_string
- price→price_decimal
- itemid→itemid_string
- prodid→prodid_decimal
- quantity→quantity_positiveInteger

3) Test data generation for input element from the input element type model

The steps for test data generation for each input element from the input element type model are as follow:

Step 1: Obtain the test data set for each node in B.

```
Algorithm: Obtaining relationships among nodes in the input element type model
Input: Input element type model T = <N, D, $n_r$, IC, RC, EE, ED>.
Output: The set R of the relationships among nodes in the model T.
R=∅;
FOR each node 'm' in N
   We assume that the outgoing edges from 'n' have
   $e_1(m,x_1,n_1)$, $e_2(m,x_2,n_2)$, ……, $e_i(m,x_i,n_i)$, where i=1;
   temp =∅;
   If $e_1$, $e_2$, ……$e_i \in$ EE then let j=1;
   FOR each edge $e_j(n,x_j,m_j)$
       We obtain the value of the facet maxOccurs and minOccurs from the $x_j$ ;
       k= random(minOccurs, maxOccurs);
       temp=temp + k*$m_j$ ;
       j=j+1;
   ENDFOR
   IF the node 'm' is labeled with the key word 'sequence' THEN
       R=R ∪m→sequence(temp)} ;
   ELSE IF the node 'm' is labeled with the key word 'all' THEN
       R=R∪{ m→all(temp) };
   ELSE IF the node 'm' is labeled with the key word 'choice' THEN
             R=R∪m→choice(temp) };
         ELSE
             R=R ∪ { m→temp };
         ENDIF
      ENDIF
    ENDIF
ENDFOR
FOR each edge e(m, x, n) in EC
     R=R ∪{m→n } ;
ENDFOR
return R;
```

Figure 5. Deriving relationships among nodes in the model.

Step 2: Obtain the relationships among nodes in T.

Step 3: Generate test data set m_TD for the node 'm' in $N \cup S \cup n_r$. We assume that we have obtained the test data sets for the node 'n_k' and the relationship $m \rightarrow key(i_1*n_1+i_2*n_2+\ldots\ldots+i_j*n_j)$ or $m \rightarrow i_1*n_1+i_2*n_2+\ldots\ldots+i_j*n_j$ is valid, where $1 \le k \le j$, $1 \le i_k$, the key is the *sequence* , *choice* or *all* , and m, $n_k \in N \cup D \cup n_r$. For the node '$n_k$', we assume that n_k_TD is the test data set for node 'n_k' and $i_k * n_k\_TD$ is the test data set for the occurrence of i_k times for the node 'n_k'. Then, $i_k * n_k\_TD$ is the Cartesian product of the test data for the node 'n_k' for i_k times. Thus, $i_k * n_k\_TD = n_k\_TD_1 \times n_k\_TD_2 \times \ldots \times n_k\_TD_{i_k}$.

- If $m \rightarrow i_1 * n_1 + i_2 * n_2 + \ldots + i_j * n_j$ or $m \rightarrow \text{sequence}(i_1 * n_1 + i_2 * n_2 + \ldots + i_j * n_j)$ or $m \rightarrow \text{all}(i_1 * n_1 + i_2 * n_2 + \ldots + i_j * n_j)$ then, $m\_TD \rightarrow i_1 * n_1\_TD \times i_2 * n_2\_TD \times \ldots \times i_j * n_j\_TD$. Considering the key word *all*, it means that its child elements may appear in any order. All of the combinations have the same possibility of revealing faults, so we consider only one of them, as a representative.
- If $m \rightarrow \text{choice}(i_1 * n_1 + i_2 * n_2 + \ldots + i_j * n_j)$, Then, $m\_TD = i_k * n_k\_TD,\ 1 \le k \le j$.

Step 4: Repeat Step 3 from the bottom up for the model T. Finally, we could obtain the n_r_TD which is the test data set for input element.

By the steps for test data generation for input element OrderRequest, we obtain 4096 test data. Three students develop three order web service versions. We apply the generated test data in them and the errors discovered are listed in table 3.

TABLE III. TEST RESULTS FOR ORDER

Version	Test Results		
	Number of faults found	*Error*	*Total execution time(sec.)*
Version1	0	-	745
Version2	1	Ignores the price which should be a positive integer	
Version3	1	Ignores the fractionDigits of pirce value which can be less than "1".	

IV. CONCLUSIONS AND FUTURE WORK

WSDL specification-based, this paper has introduced an automated test data generation approach for Web Service single operation. Both application builder and broker could apply the approach to generate test data to check whether Web Service satisfy their requirements. Tester can automatically generate black box testing use cases. It could effectively reduce the labor intensity of test and improve the degree of automation for generating Web services test data.

We are currently working on the automated tool to support our approach. The tool support to preprocess the WSDL, build the input element type models and generate test data from the model for the single operation. In addition, it permits user to select one block-box testing strategy and build the knowledge bases little by little. In the future, we will apply our approach in more projects and evaluate which one of boundary value testing, partition testing and equivalence class testing can defect more faults in Web Services.

Testing Web Service is expensive because for every test data we must send SOAP message to invoke Web Service under test and this message uses a lot of resources. Our future research will try to minimize the test data using some combinatorial approaches, such as AETG methodology [10], and minimize service invocation by using monitoring data [11].

REFERENCES

[1] W3C, "Web Services Description Language (WSDL) Version 2.0 Part 0: Primer" , W3C Recom-mendation ,26 June 2007, http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626.

[2] W3C, " Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language" , W3C Recommendation ,26 June 2007, http://www.w3.org/TR/2007/REC-wsdl20-20070626.

[3] W3C, "XML Schema Part 1: Structures Second Edition", W3C Recommendation, 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.

[4] W3C, "XML Schema Part2: Datatypes Second Edition", W3C Recommendation, 28 Oct.2004, http://www.w3.org/TR /xmlsche-2/.

[5] X. Bai, W. Dong, W. Tsai, and Y. Chen, "WSDL-Based Automatic test case generation for Web Services testing", Proseedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering (SOSE'05),20-21 Oct. 2005, China, pp.215-220.

[6] Y. Jiang, "A Method of Automated Test Data Generation for Web Service", Chinese journal of computers, Vol. 28 No. 4 Apr. 2005.

[7] J. Offutt and W. Xu, "Generating Test Cases for Web Services Using Data perturbation", ACM SIGSOFT, Software Eng. Notes, Vol. 29(5), Sep. 2004, pp. 1-10.

[8] W. Xu, J. Offutt, "Testing Web Services by XML perturbation", Software Reliability Engineering, 2005. ISSRE 2005. 16th IEEE International Symposium on, 8-11 Nov. 2005, pp.1-10.

[9] Hanna Samer ,Munro Malcolm, "An Approach for Specification-based Test Case Generation for Web Services" , Computer Systems and Applications, 2007. AICCSA '07. IEEE/ACS International Conference on ,13-16 May 2007, pp: 16-23.

[10] D. Cohen, S.R.Dalal, M.L.Fredman, and G.C.Patton, "The AETG system: an approach to testing beased on combinatorial Design", IEEE Transactions On Software Engineering, 23(7), July 1997.

[11] Massimiliano Di Penta etc, Test and Analysis of Web Services , Publisher: Springer Berlin Heidelberg, Date: 2007.

[12] Brenner, Atkinson, "Strategies for the Run-Time Testing of Third Party Web Services" , IEEE International Conference on Service-Oriented Computing and Applications (SOCA '07) pp. 114-121.