Automatic Generation of Test Data for XML Schema-based Testing of Web Services

Dessislava Petrova-Antonova[1], Kunka Kuncheva[1] and Sylvia Ilieva[2]
[1]*Department of Software Engineering, FMI, Sofia University, Sofia, Bulgaria*
[2]*Institute of Information and Communication Technologies, BAS, Sofia, Bulgaria*

Keywords: Test Data Generation, Test Input Automation, Web Services, WSDL, WS-BPEL, XML Schema.

Abstract: Service-Oriented Architecture (SOA) is a widely accepted paradigm for development of distributed applications using interoperable and flexible software components. Still the preferred technology for SOA implementation is provided by the web services. Their interface as well as complex interactions are described with XML-based standards, such as Web Service Description Language (WSDL) and Business Process Execution Language for Web Services (WS-BPEL). The WSDL and WS-BPEL documents allow automation of test data generation through instantiation of the referenced XML Schemas. The approach proposed in this paper is a step towards such goal. It provides derivation of XML instances from a given XML Schema. The approach is automated in a software tool supporting data-driven test definition. The tool automatically extracts an XML Schema form a WSDL or WS-BPEL documents and generates XML messages needed for web service interactions. Since the proposed approach supports generation of both correct and incorrect XML instances, the tool is applicable to functional as well as robustness testing of web services.

1 INTRODUCTION

Web services are the preferred way for implementation of Service-Oriented Architecture (SOA). They follow a message-enabled design that can be found on the Web and in the enterprise software (W3C, 2015). Since the web services are supposed to be platform independent, they are described through formally machine readable interfaces using Web Service Description Language (WSDL). WSDL interfaces define simple interactions with web services by using standard protocols. The web services interaction model is stateless and supports single request-response or one-way communication. The full potential of SOA is achievable when applications are able to integrate their complex interactions following a standard business process model. Such model should support long-running transactions involving two or more parties. Business Process Execution Language for Web Services (WS-BPEL) provides definition of complex business interactions including multiple units of work, often performed by different web services.

Both WSDL and WS-BPEL recognize the need of rich type system to describe the format of messages exchanged during interactions. Thus they support the XML Schemas specification as their unified type system. The XML Schemas that define message formats can be defined internally within the interface description of the web services or they can be imported externally as it is usually happens when web service orchestrations are defined. The XML-based syntax of WSDL and WS-BPEL documents allows fully automated test data generation for message exchange. It is used for specification of messages for testing web services' interactions. These messages are de facto XML instances that confirm the XML Shema prescribed in the corresponding WSDL and WS-BPEL documents. Their generation requires intensive work due to variability of the XML structure and differences in the processing of individual XML elements. Additional efforts are needed for population of generated XML documents with data both valid and invalid depending of the type of testing performed. To the best of our knowledge the only works that address such issues are (Bai, Dong, Tsai, & Chen, 2005), (Sneed & Huang, 2007) and (Bartolini, Bertolino, Marchetti & Polini 2009). The first work only outlines the possible perspectives of WSDL-based testing. The second one is focused on generation of random values for the invocation parameters but does not support interactions in

Petrova-Antonova D., Kuncheva K. and Ilieva S..
Automatic Generation of Test Data for XML Schema-based Testing of Web Services.
DOI: 10.5220/0005500602770284
In *Proceedings of the 10th International Conference on Software Engineering and Applications* (ICSOFT-EA-2015), pages 277-284
ISBN: 978-989-758-114-4

document style. The last work provides automatically derivation of data instances from WSDL descriptions. The proposed software tool has some limitations related to the processing of XML elements and generation of invalid data needed for negative testing.

In this direction, we propose an approach for instantiation of XML documents from XML Schema. The approach leverage the potential of XML Schema in describing input data in standardized manner. In addition it is realized as a software tool for automated generation of test inputs for single web services as well as web service orchestrations, described with WS-BPEL. The tool is integrated in a framework, named Testing as a Service Software Architecture – TASSA (Pavlov, Borisov, Ilieva & Petrova-Antonova, 2010). The main goal of TASSA is to support the testing of web service compositions at design time.

The paper is structured as follows. In the next section the current software tools for XML Schema-based test data generation are analyzed. Section 3 describes an approach for XML Schema-based test data generation. Section 4 presents a proof-of-concept of the proposed approach through sample usage scenarios. Conclusions are drawn in Section 5.

2 RELATED WORK

The goal of the proposed approach is to automate test data generation while performing testing of WSDL-based web services and web service orchestrations, described with WS-BPEL. That is why software tools that provide similar functionality are investigated and compared.

XML Editor (Eclipse.org) is an open source Java plugin for Eclipse that supports a single XML instance generation from an XML Schema. It generates the structure of an XML document and could fill some dummy values, but actually is designed with presumption that the user will fill them. *XML Spear* (Donkeydevelopment.com) is a free XML editor written in Java that generates an XML file from an XSD file. It is possible to configure some properties for the generated structure of the XML, but the values should be filled manually. *XML-XIG* is an open source Java application, presented by Simon Tuffs (2004) that produces more than one XML instance form a single schema and automatically generates values, based on an XIG meta-data file. *CAMed* proposed by Sorens (2009) is an XML editor implemented as an open source Java application that also produces more than one XML instance from an XSL Schema. In a template file a variety of configurations could be done including schema structure and values specification. *WS-TAXI* (Bartolini, Bertolino, Marchetti & Polini, 2009) is a free Web service, for derivation of XML instances from XML schema. It implements a category partitioning for the <choice> elements. The generated XML instances can be populated with random values or values loaded from a file. *XMLBeans* (Apache.org) is an Apache licensed set of Java classes and console tools, some of which support XML instantiation from XML Schema and validation according to that schema. *DataGen* tool, presented by Herrmann (2005) is an open source Java application that produces test cases in a XML format from an XML Schema. It provides a lot of functions especially for negative testing configured in an instruction file. The XML values could be randomly generated or predefined. *Databene Benerator*, described by Frank, Crescenzio and Chavez is a Java written tool for generation of XML instances from XSD Schema. It handles very well the generation of values, but has some disadvantages on building of the XML structure. Table 1 shows a comparison of the tools according to the following characteristics:

1) Extraction of XSD Schema from file – WSDL, BPEL, etc.;
2) Processing more than one root element;
3) Processing optional elements;
4) Processing optional attributes;
5) Processing choice elements;
6) Processing repeating elements;
7) Validation of XML files;
8) Automatic generation of valid data;
9) Automatic generation of invalid data;
10) Loading data from an external source.

Table 1: Tools for test data generation from XML Schema.

Tool	1)	2)	3)	4)	5)	6)	7)	8)	9)	10)
XML Editor		√	√	√	√	√	√			
XMLSpear		√	√	√		√	√			
XML-XIG		√					√	√		
CAMed		√	√	√	√	√	√	√	√	√
WS-TAXI	√				√		√	√		√
XMLBeans		√					√			
DataGen		√					√	√	√	
Benerator		√					√	√		√

As can be seen from the Table 1, the current software tools work only with *.xsd files and do not support extraction of XML schema from external sources, where it is referenced or embedded. Only two of them, namely CAMed and DataGen, provide generation of XML instances with incorrect data. In addition, specific XML schema elements and attributes are not handled by tools such as WS-TAXI,

Benerator, XML-XIG and XMLBeans. In order to fill these gaps, this paper proposes an approach for XML Schema-based test data generation and corresponding software tool that are described in the next sections.

3 APPROACH DESCRIPTION

This section presents an approach for generation of XML documents from a given XML Schema that is applicable to testing web services. Generally, the approach consists in sequential processing of the elements in the XML Schema file regarding their type and constraints.

Fig. 1 shows the activities covered by the proposed approach. They are divided in two main groups:

- Group 1: Activities related to the structure of the XML document or those that can be performed before its actual generation;
- Group 2: Activities performed after successively processing of the XML Schema's elements.

The order in which the activities are performed is essential since some constrains defined by the XML Schema are related to the structure of the XML document and others – to the values of the XML elements themselves.

The activities in Group 1 should be performed in the following order:

- Derivation of XML Schema from file (WSDL or WS-BPEL);
- Definition of instances' number;
- Selection of working mode;
- Extraction of *root* elements;
- Processing of elements;
- Processing of elements' attributes;
- Generation of empty XML document.

The activities included in Group 2 are related to generation of values for all elements of the XML document derived after performing Group 1 activities:

- Generation of random invalid values;
- Generation of random valid values;
- Manual definition of values;
- Substitution of values from file.

Derivation of XML Schema: Derivation of the XML Schema from file is useful when the approach is applied to web services testing. Actually, it can be used for other purposes, since it is working when an arbitrary XSD file is provided. If the XML Schema is referred in the WSDL or BPEL file, it is obtained from its physical location. In case of WSDL file, the proposed algorithm searches for *schemaLocation*

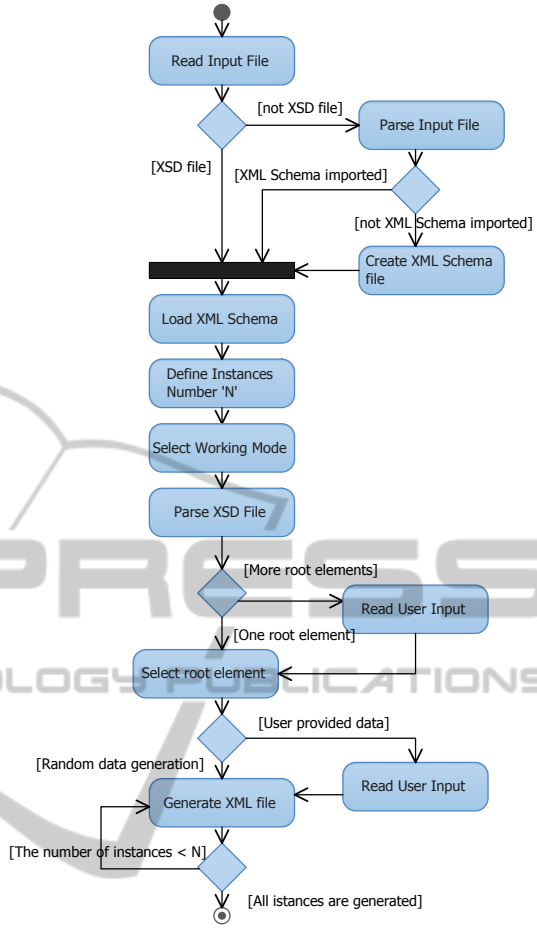

Figure 1: Activity diagram of the approach's activities.

attribute of the WSDL element *import*. If the XML Schema is inline, it uses the WSDL element *types* in order to find the definitions for the data types of the web service messages and generate XSD file for processing. When a BPEL file is provided, the algorithm uses the *location* attribute of the BPEL element *import* to obtain the XML Schema.

Definition of instances' number: The number of instances that should be generated is important, since it affects the processing of the attributes defining occurrences of the elements i.e. the number of times an element can occur in the parent element.

Selection of working mode: The working mode determines whether and how the XML instances will be populated with data. The proposed algorithm allows generation of empty XML documents – XML instances which structure follow a given XML Schema, but does not contain concrete values for the XML elements. The second and third working modes allows generation of XML instances automatically populated with random data – invalid or valid with respect of XML Schema. Thus, the approach is

applicable to negative testing when messages with corrupted data are sent to the web service under test. The fourth and fifth working modes provide options for manual specification of values or their loading from file respectively.

Extraction of *root* elements: The XML Schema may include more than one *root* elements. Their extraction is essential since further generation of the XML instance is performed for a particular root element.

Processing of elements: The elements' processing is based on their type. The elements of complex types require additional analysis of their particles. The further processing depends on whether model group, model group's declaration or element's declaration is examined. The three model groups supported by XML Schema are: *sequence*, *choice* and *all*. For example, in case of *choice* element, the child element used in the generated XML instance is selected randomly.

Processing of elements' attributes: The elements' attributes define their number of occurrences, default and fixed values. The attribute *minOccurs* specifies the minimum occurrence, and the attribute *maxOccurs* specifies the maximum occurrence. The *default* and *fixed* attributes define respectively the default and fixed value for a given element.

Generation of empty XML document: The empty XML document follow the structure prescribed in the XML Schema without concrete data for the XML elements.

Generation of random invalid values for the elements: The invalid values of the XML elements do not confirm the data types defined in the XML Schema. For example, if date type is required, the corresponding XML element receives "FAKE_DATE" value. The numeric values are represented with strings and the strings are generated with incorrect length. In case of enumeration, the XML element get value that does not exist within the enumeration. In addition, if *fixed* attribute is defined, it is neglected.

Generation of random valid values for the elements: The valid values of the XML elements confirm the data types and constraints defined in the XML Schema. If *default* or *fixed* attribute is defined, its value is assigned to the corresponding XML element. The numeric and string values are generated in accordance to restrictions, if any. If date type is required, the current date is used. The XML elements of Boolean type receive value *True*.

Manual definition of values for the elements: In case of manual definition, the user is allowed to enter a specific value for each XML element. Here, validation of data types is required.

Substitution of values from file: This working mode is considered to provide data-driven testing.

4 APPROACH FISIBILITY

This section presents a proof of concept of the proposed approach. Its feasibility is validated through implementation of software tool automating activities presented in Section 3 and subsequent execution of several usage scenarios.

Table 2: Usage scenarios for approach validation.

Usage scenario	Description
US1	Input: XSD file with more than one root element Number of instances: 2 Working mode: Generation of XML file with valid values Processing of elements' attributes: minOccurs and maxOccurs
US2	Input: XSD file with one root element Number of instances: 2 Working mode: Generation of XML file with valid values Processing of elements: choice element with 7 child element Processing of elements' attributes: minOccurs and maxOccurs
US3	Input: XSD file Number of instances: 1 Working mode: Generation of XML file with wrong values Processing of elements' attributes: fixed
US4	Input: XSD file Number of instances: 2 Working mode: Generation of XML file with valid values Processing of elements' attributes: minInclusive and maxInclusive
US5	Input: WSDL file Number of instances: 1 Working mode: Generation of empty XML file
US6	Input: WSDL file Number of instances: 1 Working mode: Generation of empty XML file and manual data population
US7	Input: WSDL file Number of instances: 2 Working mode: Generation of XML file with valid values
US8	Input: BPEL file Number of instances: 1 Working mode: Generation of empty XML file and data population from file

The usage scenarios are described in Table 2 and generally cover the following cases:

- Taking appropriate input: XSD, WSDL or BPEL file;
- Derivation of XML schema;
- Support of working modes;
- Processing XML elements and attributes.

The test cases are described in Table 2.

The XSD file passed as input to the tool for execution of Usage scenario 1 is presented in Fig. 2.

ORDER.XSD

```
<xsd:schema xmlns:xsd="..." xmlns:soap="..."
xmlns:tns="..." xmlns:wsdl="..." xmlns:xs="..."
attributeFormDefault="unqualified"
elementFormDefault="unqualified"> ...
<xsd:element name="shipto">
  <xsd:complexType><xsd:sequence>
     <xsd:element ref="name" /> ...
  </xsd:sequence>
  </xsd:complexType></xsd:element>
<xsd:element name="item">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="title" />
      <xsd:element minOccurs="0" ref="note" />
      <xsd:element ref="quantity" />
      <xsd:element ref="price" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:element name="shiporder">
  <xsd:complexType><xsd:sequence>
      <xsd:element ref="orderperson" /> ...
  </xsd:complexType></xsd:element>
</xsd:schema>
```

Figure 2: The input file for execution of Usage scenario 1.

ORDER1.XML

```
    <?xml version="1.0" encoding="UTF-8"?>
<item xmlns:xsi="..." xsi:schemaLocation="...">
  <title>string</title>
  <note>string</note>
  <quantity>1308834436</quantity>
  <price>658001404.14517</price>
</item>
```

ORDER2.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<item xmlns:xsi="..." xsi:schemaLocation="...">
  <title>string</title>
  <quantity>2039057581</quantity>
  <price>22910786.29806</price>
</item>
```

Figure 3: The output of execution of Usage scenario 1.

The file presented in Fig. 2 has three root elements, namely *"shipto"*, *"item"* and *"shiporder"*. The selected root element is *"item"*. The output of the tool is shown in Fig. 3. Each XML instance has reference to the XML Schema (order.xsd), from which it is instantiated. Since the element *"note"* has *minOccurs* attribute set to zero, it is missing in the second XML instance (order2.xml). All elements are populated with valid data according to the referenced XML Schema.

The XSD file passed as input to the tool for execution of Usage scenario 2 is presented in Fig. 4.

PURCHASEFORM.XSD

```
<xsd:schema xmlns:xsd="...">
<xsd:element name="purchaseForm">
<xsd:complexType><xsd:all>
    <xsd:element name="address">
    <xsd:complexType><xsd:choice>
     <xsd:element                    name="addr1"
type="xsd:string"/>
      ...
     <xsd:element                    name="addr7"
type="xsd:string"/>
    </xsd:choice></xsd:complexType>
    </xsd:element>
    <xsd:element name="order" type="POType"/>
  </xsd:all></xsd:complexType>
</xsd:element>
<xsd:complexType name="POType">
  <xsd:sequence>
   <xsd:element                    name="product"
type="xsd:string"/>
    <xsd:element name="quantity"
                type="xsd:positiveInteger"
                minOccurs="2" maxOccurs="3"/>
    <xsd:element                    name="price"
type="xsd:decimal"/>
```

Figure 4: The input for execution of Usage scenario 2.

The file, presented in Fig. 4 contains one *choice* element, named *"address"* with 7 child elements, named *"addr1"*, *"addr2"*, etc. The attribute *minOccurs* is used for definition of two elements, named *"quantity"* and *"shipDate"*. The output of the tool is shown in Fig. 5. Each XML instance has reference to the XML Schema (purchaceForm.xsd), from which it is instantiated. The two XML instances (purchaceForm1.xml and purchaceForm2.xml) are created using different child elements of the *"address"* element – *"addr2"* and *"addr1"* respectively. The *quantity* element appears three times in the first XML instance and two times in the second XML instance, since its *minOccurs* attribute

is set to value of 2 and *maxOccurs* – to value of 3. All elements are populated with valid data according to the referenced XML Schema.

```
PURCHASEFORM1.XML
<?xml version="1.0" encoding="UTF-8"?>
<purchaseForm xmlns:xsi="..."
xsi:schemaLocation="purchaseForm.xsd">
  <address><addr2>string</addr2></address>
  <order>
    <product>string</product>
    <quantity>2070341001</quantity>
    <quantity>497600509</quantity>
    <quantity>1600665101</quantity>
    <price>1352899985.30249</price>
    <date>Thu Oct 16 02:32:27 EEST 2014</date>
  </order>
</purchaseForm>

PURCHASEFORM2.XML
<?xml version="1.0" encoding="UTF-8"?>
<purchaseForm xmlns:xsi="..."
xsi:schemaLocation=" purchaseForm.xsd">
  <address><addr1>string</addr1></address>
  <order>
    <productName>string</productName>
    <quantity>1632261470</quantity>
    <quantity>186209829</quantity>
    <price>1992219919.18939</price>
  </order>
```

Figure 5: The output of execution of Usage scenario 2.

The XSD file passed as input to the tool for execution of Usage scenario 3 is presented in Fig. 6.

```
COLOR_FIXED.XSD
<xsd:schema   xmlns:xsd="..."   xmlns:soap="..."
xmlns:tns="..." xmlns:wsdl=".../"
attributeFormDefault="unqualified"
elementFormDefault="unqualified"
targetNamespace="http://www.example.org/1/">
  <xsd:element name="color"
               type="xsd:string" default="red"/>
</xsd:schema>
```

Figure 6: The input for execution of Usage scenario 3.

```
COLOR_FIXED1_NEGATIVE.XML
<?xml version="1.0" encoding="UTF-8"?>
<color
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="color_fixed.xsd"
>
```

Figure 7: The output of execution of Usage scenario 3.

It has only one element, named *"color"*, which *default* attribute is set to value of *"red"*.

The output of the tool is shown in Fig. 7. The XML instance is generated with wrong value for the *"color"* element.

The XSD file passed as input to the tool for execution of Usage scenario 4 is presented in Fig. 8. It has only one element, named *"age"* that has restrictions to its values in the range from 0 to 120.

```
COLOR_FIXED.XSD
<xsd:schema xmlns:xsd="..." xmlns:soap="..."
xmlns:tns="..." xmlns:wsdl="..."
attributeFormDefault="unqualified"
elementFormDefault="unqualified"
targetNamespace="...">
<xsd:element name="age">
  <xsd:simpleType>
    <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0" />
      <xsd:maxInclusive value="120" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Figure 8: The input for execution of Usage scenario 4.

The output of the tool is shown in Fig. 9. The two XML instances are generated with different valid values for the *"age"* element.

```
AGE_RANGE1.XML
<?xml version="1.0" encoding="UTF-8"?>
<age xmlns:xsi="..."
xsi:noNamespaceSchemaLocation="age_range.xsd">
17</age>
AGE_RANGE2.XML
<?xml version="1.0" encoding="UTF-8"?>
<age xmlns:xsi="..."
xsi:noNamespaceSchemaLocation="age_range.xsd">
102</age>
```

Figure 9: The output of execution of Usage scenario 4.

The input files for the remaining four usage scenarios are omitted due to the limited space of the paper as well as their less significance to the obtained results. For all of them the proposed tool successfully extracts the corresponding XML Schema. During execution of Usage scenario 5 and Usage scenario 6 empty XML instances are generated. In addition the elements of the XML instance obtained from Usage scenario 6 are correctly populated with values specified through user input. The two XML instances produced by the Usage scenario 7 are successfully filled with random data generated from the tool. These XML instances are also omitted because of

their similarity regarding the results already presented.

The elements of the XML instance generated after execution of Usage scenario 8 are populated with data from a text file, which content is presented in Fig. 10. It has sample values for all XML Schema primitive data types.

```
PARAMS.TXT
boolean###CSV###false
byte###CSV###1
date###CSV###Wed Oct 15 16:07:02 EEST 2014
dateTime###CSV###Wed Oct 15 16:07:02 EEST 2014
decimal###CSV###14.00
double###CSV###0.27
float###CSV###0.06
integer###CSV###7
long###CSV###9223372036854775807
negativeInteger###CSV###-3
nonNegativeInteger###CSV###0
nonPositiveInteger###CSV###0
positiveInteger###CSV###3
short###CSV###127
string###CSV###test data
unsignedByte###CSV###1
unsignedInt###CSV###1515
unsignedLong###CSV###99999999999
unsignedShort###CSV###89</xsd:schema>
```

Figure 10: The data file for execution of Usage scenario 8.

The output of the tool is shown in Fig. 11. Since all elements, except *"total"*, are defined in the corresponding XML Schema of type string, they are populated with value of *"test data"* as it is specified in the param.txt file. In addition the element *"total"* is initialized with value of 0.27, which is the value specified in the param.txt file for the elements of type double.

```
ORDERINFO.XML
<?xml version="1.0" encoding="UTF-8"?>
<Order_Details
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="OrderInfo.xsd">
  <firstname>test data</firstname>
  <lastname>test data</lastname>
  <email>test data</email>
  <country>test data</country>
  <state>test data</state>
  <city>test data</city>
  <postalcode>test data</postalcode>
  <address>test data</address>
  <phone>test data</phone>
  <creditcardnumber>test data</creditcardnumber>
  <total>0.27</total>
  <currencycode>test data</currencycode>
</Order_Details>
```

Figure 11: The output of execution of Usage scenario 8.

The experiments show that the proposed systematic approach implemented by the software tool can provide automated correct test data generation. Covering such wide variability in the structure and the values of created XML instances requires considerable efforts in case of manual preparation.

5 CONCLUSIONS

In this paper an XML Schema-based approach for derivation of XML instances was presented. The approach is fully automated in a proof-of-concept tool that is applicable for test data generation in case of functional testing of both single and composite web services. The problem, solved by the approach and the corresponding software tool has been widely investigated in the past years. However the current solutions provide limited support for generation of test inputs. For example, they do not cover all possible combinations of the XML Schema elements or are applicable only for functional testing. In contrast, our work adopts a systematic sequence of activities that lead to more precise derivation of XML instances. In addition, the proposed approach supports generation of XML instances with wrong structure or incorrect data and thus can be applied for robustness testing.

The plan for future work is to apply the proposed approach in different application domains, such as benchmark generation, web application testing and database population. The main task will be to evaluate the approach in real case studies. The implemented software tool will be extended with additional functionality for value generation using regular expression or function.

ACKNOWLEDGEMENTS

This work is partially supported by the ERDF under agreement n. BG161PO003-1.1.06.-0003-C0001 and by the Scientific Research fund of Sofia University "St. Kliment Ohridski" under agreement n. 143/17.04.2015.

REFERENCES

W3C, *Web of Services*, viewed 9 February 2015, <http://www.w3.org/standards/webofservices/>.

Eclipse.org, *Structured text editors for markup languages*, viewed 6 January 2015, <http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.wst.xmleditor.doc.user%2Ftopics%2Ftxedttag.html>

Donkeydevelopment.com, *XML Spear*, viewed 6 January 2015, <http://www.donkeydevelopment.com/>

Simon Tuffs P 2004, *XML-XIG: XML Instance Generator*, viewed 6 January 2015, <http://xml-xig.sourceforge.net/>

Sorens M 2009, *Taking XML Validation to the Next Level: Introducing CAM*, viewed 6 January 2015, <http://www.devx.com/xml/Article/41066>.

Bartolini C, Bertolino A, Marchetti E & A.Polini 2009, WS-TAXI: a WSDL-based testing tool for Web Services, in *Proceedings of the 2nd International Conference on Software Testing, Verification, and Validation*, Denver, Colorado USA, pp. 326-335.

Apache.org 2012, *XMLBeans*, viewed 6 January 2015, <http://xmlbeans.apache.org/>.

Herrmann H 2005, *Test Data Generation Tool*, viewed 6 January 2015, <http://iwm.uni-koblenz.de/datagen/manual/DataGen_UserManual.pdf>.

Frank H, Crescenzio D, Chavez E, *Databene Benerator*, viewed 6 January 2015, <http://databene.org/databene-benerator.html>.

Pavlov V., Borisov B., Ilieva S., & Petrova-Antonova D., 2010, Framework for Testing Service Compositions, in *Proceeding of 12th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, Timisoara, Romania, September 23-26, pp. 557-560.

Bai X., Dong W., Tsai W.-T., & Chen Y., 2005. WSDL based automatic test case generation for web services testing. In Proceding of IEEE Int. Work. SOSE, Washington, USA, IEEE Comp. Soc., pp. 215-220.

Sneed H. M., Huang S., 2007. The design and use of WSDL-test: a tool for testing web services. Journal of software Maintenance and Evolution: Research and Practice, 19:297-314.