Applying Mutation Testing to XML Schemas

Ledyvânia Franzotte
ledyvania@inf.ufpr.br

Silvia Regina Vergilio
silvia@inf.ufpr.br

Computer Science Department, CP: 19081, 81531-970
Federal University of Paraná (UFPR), Brazil

Abstract

XML language is largely used by Web-based applications to exchange data among different components. XML documents, in most cases, follow a grammar or schema that describes which elements and data types are expected by the application. These schemas are "translated" from specifications written in natural language, and consequently, in this process some mistakes are usually made. Because of this, faults can be introduced in the schemas and incorrect XML documents can be validated. This work proposes the use of mutation testing to reveal faults in schemas. A set of mutation operators is introduced by considering the most common errors present in the project of schemas. A process and a testing tool to support the operators are described. By using this tool some experiments were accomplished. Results from these experiments show the applicability of the operators, as well as, their efficacy to reveal faults.

1. Introduction

XML (eXtensible Markup Language) [12] is a markup language largely used by Web-based applications to exchange information among different components. All XML documents need to be well formed and valid. A well-formed document obeys XML syntax and this can be detected using a XML checker. A valid document follows rules defined in a schema that specifies the grammar expected for the document. Examples of schemas that are generally used are: DTD [2] and XML Schema [13].

Since developers are responsible to write the schemas, some mistakes can be made and incorrect schemas can be generated. A faulty schema can validate an incorrect XML document. If such incorrect document is used, this can cause a failure in the application that uses the schema.

Due to this scenario, the use of testing techniques and criteria to reveal faults in the schemas is fundamental. There are three basic testing techniques that can be used: functional, structural and fault-based. The fault based technique and criteria consider the main faults that are generally present in the programs being tested. The Mutation Analysis criterion is fault-based and has been considered as the most efficacious in terms of the number of revealed faults [11].

The idea of the Mutation Analysis criterion [1] is to insert small modifications in the program by creating new programs named mutants. Test data must be generated to kill all the created non-equivalent mutants. To kill a mutant it is necessary to execute a test input that produces different outputs for the mutant and for the original program. If there is no test case to distinguish the mutant from the original program, this mutant is called equivalent. At the end of the process, a mutation score is obtained. It is calculated according to the number of dead mutants and number of non-equivalent mutants generated.

There are in the literature some works that explore fault-based testing in the context of Web-based applications. The works described in [5, 7, 15] introduce mutation operators with the goal of testing the interaction between Web components, mainly web-services and have a different focus. In [3, 6], the authors explore types of faults in schemas. However, the type of faults introduced in both works is not used in a typical mutation testing. Emer et al, do not generate mutant schemas. Li and Miller do no apply the criterion. Some questions are not addressed, such as: "How to produce different behaviors for the mutant and original schema?" or "How to obtain the mutation score?"

Our work explores the use of Mutation Analysis for testing schemas, in the case, XML Schema that has become a very popular type of schema for XML documents. To allow the application of this criterion a set of operators and a supporting tool are necessary. Hence, we introduce a set of mutation operators based on the type of faults described by works from the literature, mentioned above. We also propose the use of XML documents as test data to be, or not, validated by the schemas. In this way, it is possible to kill the mutants and to obtain the mutation score to evaluate test cases. A tool, named XTM (Tool for **X**ML Schema **T**esting Based on **M**utation), was implemented. By using this tool, we accomplished two experiments to evaluate the proposed operators. The results allowed comparison with works from literature.

This paper is organized as follows: Section 2 shows related work. Section 3 introduces the mutation operators.

Section 4 describes the tool XTM and how to evaluate the mutants. Section 5 presents the results from the experiments and, Section 6 contains the conclusions and future work.

2. Related Work

There are in the literature some works that explore fault-based testing in the context of web applications [3, 5, 6, 7, 15]

Lee and Offutt [5] introduce two operators: *lenghtOf* and *memberOf* for XML documents. The created mutants are used to test the communication between components of a web application.

Offutt and Xu [7] explore the use of perturbation operators to test interaction in web-services. Data value and interaction perturbations are explored. Data perturbation operators modify data values by applying the idea of boundary value analysis. Interaction perturbation addresses RPC and data communications.

Xu et al [15] propose some mutation operators to schema perturbation, in this case, using XML Schema. The schema is represented by a tree T and operators that change sub-trees and nodes of T are introduced to create incorrect XML messages used in the test of web services.

The above mentioned works do not have the goal of testing schemas. The works described in [3, 6] address the test of XML Schema documents and have similar objective to our work.

Emer et al [3] introduce classes of faults and suggest the creation of mutant XML documents to be queried with the goal of revealing faults in XML Schema. The results from these queries are confronted with the schema specification. The authors do not apply the Mutation Analysis criterion and mutant schemas are not generated. They show results about efficacy by using four schemas of two systems.

Li and Miller [6] proposed 18 mutation operators for schemas. Each operator changes a value or an attribute and was specified based on the most common faults made by developers. These operators are described in Table 1. However, the authors do not address some questions related to the application of the Mutation Analysis criterion, such as: “How to kill a mutant?” or “How to obtain the mutation score and to evaluate a test set?”. These questions are important because the goal of a testing criterion is to help the tester in the task of generation and evaluation of a test set. The authors do not evaluate the proposed operators with respect to efficacy. They only conduct an experiment to show that common used parsers do not reveal most faults described by their operators.

Table 1 – Mutation operators proposed by Li and Miller[6]

Oper.	Description
AOC	Changes the value of attribute occurrence constraint.
SPC	Changes the regular expression declared by using the facet pattern.
RAR	Changes the value range declared.
EEC	Reduces or increases the number of arguments one by one.
SLC	Changes the value of the facets: *length, minLenght* and *maxLenght.*
NCC	Changes the value of the facet *fractionDigits* and *totalDigits*.
DTC	Replaces complex type deriving type modifiers.
UCR	Replaces the control modifier.
SNC	Changes XML Schema namespace
TDE	Exchanges the XML Schema target namespace and attribute
ENR	Replaces the namespace identifier of the element.
ENM	Removes the prefixed element namespace identifier.
QCR	Replaces the definition of the qualification of local elements and attributes.
QIR	Similar to QCR, but with control qualification on a declaration-by-declaration basis using the *form* element.
CCR	Replaces the complex type compositors.
COC	Changes the order of elements one by one.
CDD	Reduces the number of elements one by one.
EOC	Changes the value of an element occurrence constraint.

By analyzing the works mentioned in this section, we observe that the operators proposed by Li and Miller need to be evaluated. They do not consider some faults described by Emer et al, as well as faults described by the operators proposed to test web services, that can be also adapted to the context of schema testing. Because of this, in next section, we introduce a new set of operators and in Section 5, we present evaluation results.

3. Mutation Operators

In this section, mutation operators to XML Schema documents are introduced. These operators were defined by considering the most common faults made by developers when writing schemas and the works described in last section.

The operators are divided in two groups that generate elementary and structural mutations. Each operator represents a possible fault and produces a simple change in the schema being tested. Each operator is illustrated with a fragment of a XML Schema document. For this fragment, possible mutations are presented.

The XML Schema structure will be represented as a tree T and, each schema element will be called node (terminal and non-terminal). Non-terminal nodes in T will also be called sub-tree.

3.1 Elementary Mutation Operators

These operators modify attributes values and elements in the document. This group contains the following operators:

- **Group_Order (GO):** changes the order in which the elements are expected to be. The operator domain is: {sequence, one, many, choice}
  Example:
  <group order=”**ONE**”>
  Possible mutations:
  <group order = “**SEQ**”>;
  <group order= “**CHOICE**”>;
  <group order= “**MANY**”>
- **Required (REQ):** changes type of the attributes, if it must be obligatory or not. The operator domain is: {yes, no}
  Example:
  <attributetype name="bar" dt:type="int" required="**YES**"/>
  Possible mutations:
  <attributetype name="bar" dt:type="int" required="**NO**"/>
- **DataTypes (DT):** changes data type of elements and attributes. The operator domain contains all types defined, for instance: {integer, string, float, decimal, int, enumaration}.
  Example:
  <attributetype name="bar" dt:type="**INT**" required="yes"/>
  Possible mutations:
  <attributetype name = "bar" dt:type="**STRING**" required="yes"/>;
  <attributetype name="bar" dt:type="**FLOAT**" required="yes"/>
- **LenghtOf (LO):** changes the name size of the elements. Similar to *lengthOf* [5].
  Example:
  <element type="**A**">
  Possible mutations:
  <element type="**AXXX**">
- **ChangeSingPlural (CSP):** changes the element size by adding or removing the char ‘s’ at the end of the string.
  Example:
  <element name="**A**">
  Possible mutations:
  <element name="**AS**">
- **ChangeTag (CTP):** changes the most common node tag used. Similar to: CCR, DTC and RAR [6].
  Example:
  <**attributetype** name="bar" dt:type="int" required="yes"/>
  Possible mutations:
  <**ELEMENTTYPE** name="bar" dt:type="int" required="yes"/>
- **SizeOccurs (SO):** changes the size of maximum and minimum occurrence, either from the fixed types (string) or from the types defined by the user. (Similar to EOC [6]).
  Example:
  <element type="B" minOccurs="**1**">
  Possible mutations:
  <element type="B" minOccurs="**0**">

3.2 Structure Mutation Operators

In this group, there are three operators to generate mutants by modifying the tree structure of the schema under test.

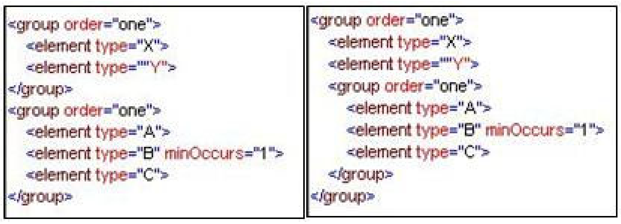
- **SubTree_Exchange (STE):** inverts the sub-trees below some node. Similar to: changeE [15] and COC [6]. Figure 1 shows a part of an XML as an example (original and mutant).

- **Insert_Tree (IT):** adds a node in the structure of the sub-tree. Similar to: insertN, insertND [15] and EEC [6]. Figure 2 shows a part of an XML as an example (original and mutant).

- **Delete_Tree (RT):** removes the node (or the sub-tree) from the structure of the tree. Similar to: deleteN, deleteND [15], CDD and EEC [6]. Figure 3 shows a part of an XML as an example (original and mutant).

4. Applying Mutation Testing

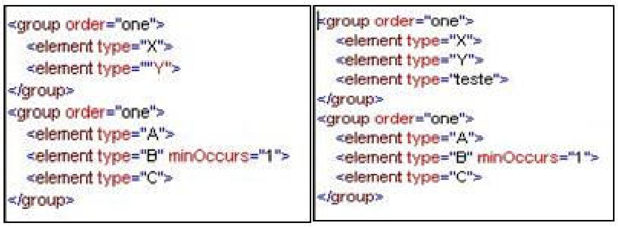
To apply the Mutation Analysis criterion and the operators introduced in last section, we propose the process presented in Figure 4. Mutation operators are used to perturb a given XML schema, and several mutants are generated. After this, the tester needs to provide a set of test data, XML documents. Each mutant schema as well as the original one are used to validate all the documents in the test set. A mutant is considered dead if the validation of a test case by using the mutant produces a different result from the validation of the same test case against the original schema.



```
<group order="one">
  <element type="X">
  <element type=""Y">
</group>
<group order="one">
  <element type="A">
  <element type="B" minOccurs="1">
  <element type="C">
</group>
```

```
<group order="one">
  <element type="X">
  <element type=""Y">
  <group order="one">
    <element type="A">
    <element type="B" minOccurs="1">
    <element type="C">
  </group>
</group>
```

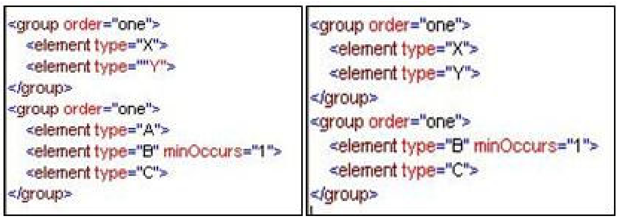
Figure 1: SubTree_Exchange example



```
<group order="one">
  <element type="X">
  <element type=""Y">
</group>
<group order="one">
  <element type="A">
  <element type="B" minOccurs="1">
  <element type="C">
</group>
```

```
<group order="one">
  <element type="X">
  <element type="Y">
  <element type="teste">
</group>
<group order="one">
  <element type="A">
  <element type="B" minOccurs="1 ">
  <element type="C">
</group>
```

Figure 2: InsertTree example



```
<group order="one">
  <element type="X">
  <element type=""Y">
</group>
<group order="one">
  <element type="A">
  <element type="B" minOccurs="1">
  <element type="C">
</group>
```

```
<group order="one">
  <element type="X">
  <element type="Y">
</group>
<group order="one">
  <element type="B" minOccurs="1">
  <element type="C">
</group>
```

Figure 3: Delete_Tree example

At the end, the mutation score is obtained for the test set provided by the tester. The tester can use the mutation score to stop testing or to compare different test sets. If necessary, additional XML documents can be used to improve the score and to kill the remaining alive mutants. Among these mutants, there can be some ones that are equivalent to the original schema. The tester needs to determine the equivalence of schemas.

XTM (Tool for **X**ML Schema **T**esting Based on **M**utation) is a tool that supports the described process. It was developed in JAVA language using JDOM API [4] to read and manipulate the schemas. During the tree reading, the operators are applied and the generated mutants are validated using SAX API [8]. These schemas are used to validate the XML documents, used as test data.

XTM supports all the operators described in last section: GO, REQ, DT, LO, CSP, CTP, SO, STE, IT, RT. To allow comparisons with the work of Li and Miller [6], the following complementary operators were also implemented: SNC, QCR, AOC, QIR, CCR, UCR, SLC, RAR, NCC, DTC, SPC. The operator CTP was not individually implemented. The mutants generated by CTP are included in the set of mutants generated by CCR, DTC and RAR.

5. Experiments

We conducted two experiments to evaluate the proposed operators and the testing process implemented by XTM. To allow comparison with the approach proposed by Emer et al., we used the same schemas and systems mentioned in [3]: Enroll system (schemas: student.xsd (E1) and discipline.xsd (E2)) and Library system (schemas: books.xsd (E3) and users.xsd (E4)). These schemas, however, are simple. Because of this, we also use the schemas sia.xsd (E5) and sih.xsd (E6) from the real system of a hospital. Table 2 shows, for each schema, the total number of mutants generated by each operator implemented by XTM.

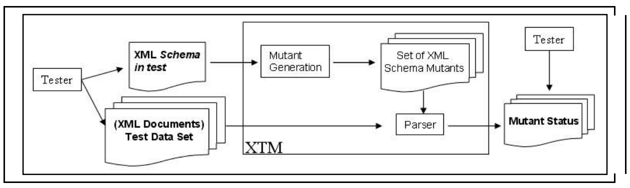


Figure 4: Using the Mutation Operators

The first experiment, also conducted by Li and Miller, shows that the most faults described by the operators produce valid schemas and are not detected by most parsers. The second one evaluates the operators and the testing process, according to cost and efficacy.

5.1 Experiment 1

The goal of this experiment is to show that usual schema parsers are not able to identify the faults described by the proposed operators. To do this, the same methodology described in [6] was used. All the generated mutants were submitted to the following parsers: W3C [14], XMLSpy [9] e Stylus Studio [10]. As a result of this step, the mean percentage of valid mutants was obtained by considering all the parsers. Table 3 shows these percentages for each schema and operator. In this table, the results of the elementary mutation operators (Group 1) are grouped in column G1.

The total (Tot1) presented in Table 2 does not consider the document validation. The total (Tot2) presents the number of valid mutants according to W3C [14] parser. In this table, the first ten operators were proposed by Li and Miller [6] and the other ones were proposed in this work. Observe that, it was not possible to apply some operators proposed by Li and Miller to all the schemas. According to Table 2, most mutants were generated with the operators proposed in this work. This is due to the structure of the used schemas, that are simple. A better analysis should be done to verify the impact of these operators and evaluate the frequency of use of these structures in the schemas.

Considering Table 3, we observe that around 60% of the mutants are valid. This means that the major faults they represent are not revealed by the parsers commonly used to check XML documents.

The InsertTree operator generated a large number of invalid mutants due to its implementation characteristics. A node is inserted without analysis of its use context, generating, in this way, several invalid mutants. This operator needs improvement to avoid this.

Table 2: Mutants Generated by Mutation Operator

Oper.	**XML Schema**						**Tot1**	**Tot2**
	E1	**E2**	**E3**	**E4**	**E5**	**E6**		
SNC	1	1	1	1	1	1	6	0
QCR	1	1	1	1	1	1	6	6
AOC	0	0	0	0	0	0	0	0
QIR	0	0	0	0	0	0	0	0
CCR	4	4	6	4	16	16	50	50
UCR	0	0	0	0	0	0	0	0
SLC	0	0	0	0	0	0	0	0
RAR	0	0	0	0	3	3	6	6
NCC	0	0	0	0	0	0	0	0
DTC	0	0	0	0	1	1	2	2
SPC	0	0	0	0	0	0	0	0
GO	0	0	0	0	0	0	0	0
CSP	10	10	11	8	36	34	109	107
REQ	0	0	0	0	0	0	0	0
DT	12	12	10	8	52	48	142	139
SO	23	20	32	25	80	75	255	218
LO	10	10	11	8	35	36	110	110
IT	22	28	28	20	80	83	261	62
STE	28	28	28	15	77	96	271	268
RT	4	15	15	12	54	57	157	153
Total	115	129	143	102	436	451	1376	1021

5.2 Experiment II

The goal of the proposed operators is to describe faults related to semantic and not just syntactic faults. As shown in Experiment I, the faults are not revealed by just using parsers. To reveal the faults described by the operators it is necessary to apply the mutation testing, which was done in a second experiment, named Experiment II.

For each schema, only the mutants that were validated by the parser implemented by XTM were used (mechanism using SAX API [8]). Table 4 presents the number of valid mutants generated for each schema, number of necessary test cases to kill them and the obtained score. All the generated mutants were dead (Score=1), that means that it was not generated equivalent mutants.

To evaluate the efficacy of the operators, we considered the same faults revealed by the approach proposed by Emer et al [3] for schemas E1, E2, E3 and E4. For this experiment, a fault was seeded in the schema E5. The number of known and revealed faults is also presented in Table 4.

Table 3: Percentage of Valid Mutants

XML Schema	**% Valid Mutants Media**				
	G1	**SBT**	**IT**	**RT**	**Total**
E1	60.10	100	32,14	100	68,24
E2	68.39	100	14,29	100	72,83
E3	68.12	100	14,29	100	71,97
E4	67.23	100	15	100	69,15
E5	65.57	100	12,50	96.3	69,62
E6	59.43	100	14,46	96,49	74,62
Total	64.80	100	17,11	98,8	71,02

Table 4: Results from Experiment II

XML Schemas	**Valid Mutants**	**Score**	**Number of Test Cases**	**Revealed/ Known Faults**
E1	123	1	3	10/10
E2	120	1	4	8/8
E3	133	1	3	4/4
E4	100	1	3	9/9
E5	420	1	4	0/0
E6	455	1	4	1/1

The type of revealed faults of Enroll and Library systems were about: size, patterns and fields data type, like login and password, occurrence size and limit values allowed. The fault of Schema E6 is related to the limit of a created data type.

We can observe, that all the known faults could be detected by the test cases. The same faults revealed by the approach of Emer et al. were discovered with a total of 21 test cases. Emer et al. use 22 queries for testing only schema E1. Beside this, 31 faults, out 32, were discovered by using the operators proposed in this paper. The implemented operators proposed by Li and Miller [6] revealed only one fault presented on E6. This fault was detected by the operator RAR and it is the same fault not detected by the operators proposed in this work.

6. Conclusions

In this work, it was defined a testing process for XML Schema documents based on the Mutation Analysis criterion. A set of mutation operators for XML Schema was proposed to generate diverse mutant schemas. XML documents are used as test data to kill the mutants. If a mutant validates a document and the original schema does not, or if the mutant does not validate the document, but the original schema does, the mutant schema is considered dead. In this way, the mutation score can be calculated to evaluate the testing activity or to compare test sets.

The proposed operators modify the schema under test by considering two dimensions: data and structure. They are based on typical faults that can be generally present in the schemas.

To allow the application of the criterion, a tool, named XTM was implemented. This tool supports the introduced operators and other ones found in the literature. By using this tool two experiments were accomplished.

The first experiment shows that the most faults described by the operators (around 60%) were not detected by using parsers, commonly used to check XML documents. These faults when present in the schemas allow that incorrect XML documents are exchanged and this can cause a failure in the applications.

In the second experiment, the operators implemented by XTM revealed all the known faults. We observed that the operators proposed in this paper are capable of revealing a great number of faults and can be considered as complementary to the ones proposed in works from the literature.

The application of mutation testing can be an expensive process. It is directly proportional to the size and to the complexity of the schema under test. The use of a tool like XTM is fundamental, as well as to choose the operators to be applied. We also observe that some operators did not generate any mutant. We are now implementing some mechanisms in XTM to enable operators according to the characteristics of the schema being tested. This helps to decrease the number of generated mutants. Other experiments should be conducted to better evaluate these characteristics and the operators.

The proposed operators should be improved and new ones should be proposed. The operator InsertTree needs to be redefined to consider the context in which the insertion should be made. In addition, these operators can also be used in other contexts, such as the communication test of Web Services.

References

[1] DEMILLO, R. A; LIPTON, R. J. Hints on test data selection: help for practicing programmer. IEEE Computer, v. 11, n. 4, p. 34-41, April, 1978.

[2] W3C. Document Type Definition. Available: http://www.w3schools.com/dtd/default.asp. Access: February, 2005.

[3] EMER, M. C. F. P. and VERGILIO, S. R. and JINO, M. A Testing Approach for XML Schemas. In: The 29th Annual International Computer Software and Applications Conference, COMPSAC 2005 - QATWBA 2005, July, 2005.

[4] JDOM, JAVA DOM. Available: http://www.jdom.org/. Access: August, 2005.

[5] LEE, Suet Chun. OFFUTT, Jeff, Generating Test Cases for XML-based Web Component Interaction Using Mutation Analysis. In: Proceedings of the 12th International Symposium o Software Reliability Engineering, p. 200-209, Hong Kong, China, November, 2001.

[6] LI, Jian Bing, MILLER, James. Testing the Semantics of W3C XML Schema. In: he 29th Annual International Computer Software and Applications Conference, COMPSAC 2005 - QATWBA 2005, July, 2005.

[7] OFFUTT, J. and XU, W. Generating Test Cases for Web Services Using Data Perturbation. In: TAV-WEB Proceedings, September, 2004.

[8] SAX Project. Available: http://www.saxproject.org/. Acess: February, 2005.

[9] XMLSpy2005. Available: http://www.altova.com/products_ide.html. Acess: November, 2005.

[10] Stylus Studio 2006. Available: http://www.stylusstudio.com/xml_download.html. Acess: November, 2005.

[11] WONG, W.E. and MATHUR, A.P. and MALDONADO, J.C. Mutation Versus All-uses: An Empirical Evaluation of Cost, Strength and Effectiveness. In: Software Quality and Productivity – Theory, Practice, Education and Training, Hong Kong, December, 1994.

[12] W3C. Extensible Markup Language (XML) 1.0 (second edition) – W3C recommendation, October 2000. Available: http://www.w3.org/XML. Access: January, 2005.

[13] W3C. XML Schema recommendation, May, 2001. http://www.w3.org/tr/. Access: January, 2005.

[14] W3C Validator for XML Schema. Available: http://www.w3.org/2001/03/webdata/xsv. Acess: November, 2005.

[15] XU, Wuzhi, OFFUTT, Jeff, LUO, Juan. Testing Web Services by XML Perturbation. In: Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering, November, 2005.