Testing Web Services by XML Perturbation*

Wuzhi Xu, Jeff Offutt and Juan Luo
Info. & Software Engineering, George Mason University
Fairfax, VA 22030-4444 USA
wxu2@gmu.edu, offutt@ise.gmu.edu & jluo2@gmu.edu

Abstract

The eXtensible Markup Language (XML) is widely used to transmit data across the Internet. XML schemas are used to define the syntax of XML messages. XML-based applications can receive messages from arbitrary applications, as long as they follow the protocol defined by the schema. A receiving application must either **validate** *XML messages,* **process** *the data in the XML message without validation, or* **modify** *the XML message to ensure that it conforms to the XML schema. A problem for developers is how well the application performs the validation, data processing, and, when necessary, transformation. This paper describes and gives examples of a method to generate tests for XML-based communication by modifying and then instantiating XML schemas. The modified schemas are based on precisely defined schema primitive perturbation operators.*

Keywords: Software testing, web services, XML

1. Introduction

HTML is intended to transmit and display electronic documents, whereas XML is intended to transmit and display data. XML allows users to define their own tags and use DTDs or schemas to specify grammars that define the syntax. Applications can then use the grammars to validate the syntax of XML messages. This does not, of course, ensure that the values in the XML are correct or that the software that uses the XML messages is correct. One common use of XML is in web services. *Web services* are software programs that operate independently to offer *services* over the Internet to other software programs, including web applications and web services. Web services usually use "peer-to-peer communication" as opposed to client-server communication, that is, data is transmitted between two essentially independent software entities. This allows a kind of *dynamic integration*, where services are found dynamically during execution. Web services use XML for specifications, configuration, and communication protocols. Web services use XML to describe and transmit data.

This paper focuses on the use of XML to transmit data, and introduces new methods to generate tests from existing XML messages. The method depends on an XML schema to describe the data format. We are also developing algorithms to *infer* a schema for cases where no schema exists.

Some applications and web services do not validate XML messages against an XML schema, and sometimes no schema exists, so validation is not possible. If an application validates an XML message and it does not conform to the schema, the application has options. It can reject the XML message, modify the XML message to conform to the schema, or process the portions of the message that are correct and ignore the rest. To evaluate the application, we need to know how well the application validates the XML messages, how well it processes the data without validating, or how effectively it transforms the messages to conform to the schema. This paper proposes a method to evaluate how effectively web services perform these tasks by using existing XML messages and schema to produce and transmit *invalid* data.

The method in this paper defines *schema perturbation operators* that are used to modify XML schema. First a formal model for XML messages and schema is developed. This model is used to algorithmically process XML messages. Perturbation operators are defined on this model, which are used to generate test inputs for the web services and applications that use the XML messages. This technique has been applied to two web service applications to evaluate its ability to find software faults.

2. Background

This section provides an overview of web services, XML, XML schemas, and some of the early attempts to test

*This work was supported in part by the U.S. National Air and Space Administration, GSFC, under contract grant 220873 to Indus Corporation, subcontracted to George Mason University. The first author acknowledges support by Avaya Research Labs. The second author is a part-time faculty researcher at the National Institute of Standards and Technology.

web services.

2.1. Web Services and XML Schemas

One difficulty that research scientists have in this area is that there are several contradictory and imprecise definitions for web services [17]. As a working definition for our research, we define a *web service* to be an Internet-based, modular application that uses the Simple Object Access Protocol (SOAP) for communication and transfers data in XML through the Internet.

SOAP allows web services to be described, advertised, discovered and invoked through the Internet. The *Extensible Markup Language (XML)* [26] is used to transmit messages and data. The *Universal Description, Discovery and Integration (UDDI)* specification is used to maintain directories of information about web services. The *Web Services Description Language (WSDL)* is used to describe how to access web services and what operations they can perform. SOAP helps software transmit and receive XML messages over the Internet.

Like HTML, XML uses tags, which are textual descriptions of data enclosed in angle brackets ('<' and '>'). Unlike HTML, XML follows strict SGML syntax. XML messages must be *well-formed*, that is, have a single document element with other elements properly nested under it, and every tag must have a corresponding closing tag. XML messages can be constrained by grammar definitions. The grammars can be defined in either Document Type Definitions (DTD) or the more recent XML schema [27], which provides stronger data typing features. Programmers can *validate* XML messages against their grammars.

2.2 Testing Web Services

One way to describe web services is that the components are wrapped with SOAP interfaces so they can exchange XML-based messages. This description is simple and reasonably accurate, but it masks some of the complexities. To consider their complexities, we need to consider how traditional programs become web services. Aoyama describes three evolutionary ways [3]. In each, web services are often used to publish traditional software on the Internet or to integrate subsystems within an organization. Web services are more widely distributed than traditional software. The fundamental objective of using web services today is the same as that of distributed computing technologies 20 years ago: to allow applications to work cooperatively with other applications over a common network [10]. However, these three methods of software evolution highlights some differences between web services and traditional software.

Moreover, the ways the technologies are used make testing web services different from testing other distributed software. Companies use web services to reduce the cost of integrating heterogeneous subsystems across the organizations [3, 7]. Web services can be considered to be **more** heterogeneous than web applications. Web applications often use more than one programming language, but different components of web **services** also use different operating systems and different server containers [30].

Another difference is based on the fact that web services do not have user interfaces [11]. This makes testing harder by reducing the **controllability** [6] of web services but simultaneously makes it easier by forcing the inputs to be structured (with XML and SOAP). To be able to find, request, and receive "services" automatically and dynamically, they use a dynamic and loosely coupled Service-Oriented Architecture (SOA) [10]. That is, web services feature **dynamic integration**. This is different from earlier distributed computing architectures that linked applications together with static connections.

Several papers [2, 5, 12, 19, 23] have been published about testing web applications. Some existing testing techniques that are used to test software components are being extended to web services. Also, some existing testing techniques for distributed applications are used to test web services. A few papers [7, 11] have presented testing techniques for web services, but the dynamic discovery and invocation capabilities of web services bring up many testing issues that have not yet been addressed. Three main aspects of web services must be tested: (1) the discovery of web services, (2) the data format exchanged, and (3) the request/response mechanisms. Bloomberg suggests four abilities that must be tested in web services [7]: SOAP messages; WSDL files; the publish, find, and bind capabilities of an SOA; and services consumer and producer emulation.

In addition, there are several papers about modeling the composition of web services and model checking on the composed web services. Bultan, Fu et al. [8, 14] create a model for web services to describe composite web services, focusing on an the messages in web services. Finite state machines are used to represent web services that have a sequence of messages, as observed through conversations. This model consists of multiple peers that communicate with asynchronous messaging. Based on the formal model and an FSM implementation, they study the relationship of global behavior of composite web services and local behaviors of the individual web services in the position. Also based on this model, model checking is used to check the global behavior of composited web services.

The testing method in this paper focuses on peer-to-peer communication, not on compositions of web services. The currently available testing techniques [7] for peer-to-peer communication focus on testing SOAP messages, testing WSDL files, and emulating consumer and producer emulation. Most web services testing tools that test SOAP mes-

sages focus on testing RPC communications, and do not include general XML data communications. The research presented in this paper uses data perturbation to test the integration of software components that rely on **both** kinds of communications.

3. XML Data Model

To generate test data from XML, a formal model for XML is needed. Representing the structure of an XML schema is relatively straightforward, because the XML language is inherently structured, but it is not as easy to represent the constraints. Several other researchers have created formal models of XML [4, 13, 22], but none represent representation constraints, which are important to our research. Our model is based heavily on these previous models and a preliminary version was discussed in a workshop paper [20].

An XML schema can be modeled as a tree. We denote the XML tree $T = (N,\ D,\ X,\ E,\ n_r)$, where:

- N is a finite set of elements and attribute nodes.
- D is a finite set of built-in and derived data types.
- X is a finite set of constraints (integrity and representation).
- E is a finite set of edges. An edge can be expressed as $e\ (p,\ x,\ c)$, where $x \in X$, $p \in (N \cup \{n_r\})$, and $c \in (N \cup D)$
- n_r is the root node.

The integrity and representation constraints in X are as defined in the XML Schema Recommendation 1 [28]. All built-in data types defined in XML Schema Recommendation 2 [29] are included in D. We need to identify all constraints and the built-in and derived data types to use this model. The model is in the form of a tree T. If T' is a subtree of T, it can be expressed as $T' \subset T$. T' can also be expressed as an edge, $e\ (n,\ x,\ T')$, where $n \in N$ and $x \in X$. In this model, any path that begins at the root node will eventually end at a data type in the set D. This principle can be expressed as a formula in CTL [15]: given a tree $T = (N,\ D,\ X,\ E,\ n_r)$, $AG(n \rightarrow AF(d))$ must be satisfied, where $n \in N$ and $d \in D$. Consider the simplified XML schema for books shown in Example 1.

Example 1 *: Simplified XML Schema for Books*

```
<xs:element name="books">
 <xs:complexType>
 <xs:sequence>
  <xs:element name="book" maxOccurs="unbounded">
   <xs:complexType>
   <xs:sequence>
    <xs:element name="ISBN" type="xs:string"/>
    <xs:element name="price" type="priceType"/>
    <xs:element name="year" type="xs:int"/>
   </xs:sequence>
   </xs:complexType>
  </xs:element>
 </xs:sequence>
 </xs:complexType>
</xs:element>

 <xs:simpleType name="priceType">
  <xs:restriction base="xs:decimal">
   <xs:fractionDigits value="2"/>
   <xs:maxInclusive value="1000.00"/>
  </xs:restriction>
 </xs:simpleType>
</xs:schema>
```

The book XML schema is represented by the model $T = (N,\ D,\ X,\ E,\ n_r)$, where:

- $n_r = books$
- $N = \{book,\ ISBN,\ price,\ year\}$
- $D = \{string,\ decimal,\ int\}$
- $X = \{maxOccurs =$"$unbounded$", $maxInclusive =$"1000.00", $fractionDigits =$"2"$\}$
- $E = \{e_1\ (n_s, x_1,\ book),\ e_2\ (book, x_2,\ ISBN),\ e_3\ (book, x_3,\ price),\ e_4\ (book, x_4,\ year),\ e_5\ (ISBN, x_5,\ string),\ e_6\ (price, x_6,\ decimal),\ e_7\ (year, x_7,\ int)\}$, where $x_1 = (maxOccurs =$"$unbounded$"**)**, $x_{2,3,4,5,6} = \phi$, $x_7 = (maxInclusive =$"1000.00" $\wedge$ $fractionDigits =$"2"**)**

The XML schema for books includes one integrity constraint (*maxOccurs="unbounded"*) and two representation constraints (*maxInclusive="1000.00"* and *fractionDigits="2"*). Three elements can be defined (ISBN, price and year). The data structure can be expressed in a tree with seven edges, as shown in Figure 1.

4. Schema Perturbation Operators

Our next goal is to perturb the XML schemas to create invalid messages. We define a collection of schema perturbation operators to affect the modifications[1]. Some research has already been carried out on XML transformation [18, 24] to support analysis and maintenance of XML messages. XML transformation include data document and XML schema changes, and change primitives for XML transformation are either data changes or schema changes [24]. For testing, this research focuses on XML schema

[1]There is some disagreement over whether this is a variation of mutation testing or not. Mutation is customarily applied to programs, not data, and usually not used to generate tests directly. The distinction depends on whether mutation is defined to apply to programs or to general structures that are derived from grammars. For now, we choose to use different terminology.

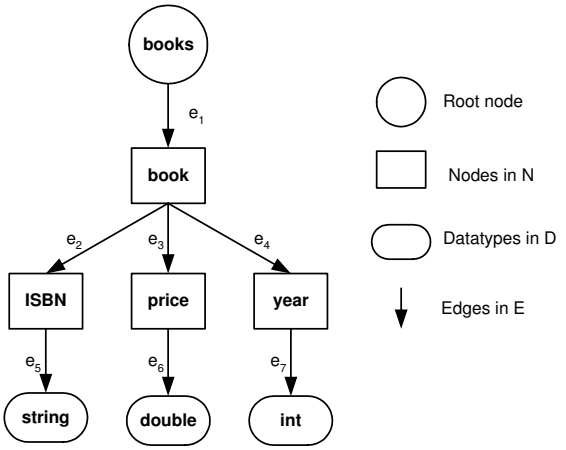

Figure 1. Tree representation of the book XML schema.

changes to produce **invalid** XML messages, and differentiates from previous research that created valid XML messages by using the term "perturbation" instead of "change." The formal model for XML schemas defines three elements in the tree: nodes, datatypes, and edges. The schema perturbation operators systematically modify these three elements. They do not perturb the root node, but can operate on trees by inserting nodes and edges into and deleting them from trees. Each perturbation operation needs a location, and some operators need an operand. The perturbation operators are formally defined with a signature, precondition, and postcondition. In the signatures, variables that will be inserted into the tree or perturbed are denoted with a single quote ('). The four perturbation primitive operators are defined as follows:

1) $insertN\ (e,\ e'_p,\ e'_c,\ n')$
Action: A new node n' is inserted between two nodes connected with the edge e. The original edge e is removed from the tree and two new edges e'_p and e'_c are inserted. Edge e'_p connects n' with the parent node, and e'_c connects n' with the child node.
Precondition: In T, $\exists\ e\ (n_p,\ x,\ n_c) \in E \wedge n_c \in N \wedge n' \in N$
Postcondition: In T', $E' = (E \setminus \{e\ (n_p,\ x,\ n_c)\}) \cup \{e'_p\ (n_p,\ x_p,\ n'),\ e'_c\ (n',\ x_c,\ n_c)\} \wedge N' = N \wedge D' = D \wedge X' = (X \setminus \{x\}) \cup \{x_p, x_c\}$

2) $deleteN\ (n)$
Action: Delete a node n from the tree; also deleting the edge between the node n and the parent node n_p. The edges between n and any child nodes n_c are moved up to become the edges between the parent node n_p and the child nodes n_c, and the constraints on the edges are unperturbed.
Precondition: In T, $\forall\ e_p\ (n_p,\ x_p,\ n)$, $e_c\ (n,\ x_c,\ n_c) \in E \wedge n_p, n_c, n \in N$
Postcondition: In T', E' = $(E \setminus \{e_p\ (n_p, x_p, n),\ e_c\ (n, x_c, n_c)\}) \cup \{e'\ (n_p, x_c, n_c)\} \wedge N'$=$N \setminus \{n\} \wedge D'$=$D \wedge X'$=$X \setminus \{x_p\}$

3) $insertND\ (n_p,\ e'_p,\ n',\ e'_c,\ d')$
Action: Insert a new node n' that connects its datatype d' with the edge e'_c below the node n_p. The edge between n_p and n' is e'_p.
Precondition: In T, $\neg\ \exists\ (e\ (n_p,\ x,\ d) \in E \wedge d \in D)$
Postcondition: In T', $E' = E \cup \{e'_p\ (n_p, x'_p, n'),\ e'_c\ (n', x'_c, d')\} \wedge N' = N \cup \{n'\} \wedge X' = X \cup \{x'_p, x'_c\} \wedge D' = D$

4) $deleteND\ (n)$
Action: Delete a node n with its datatype.
Precondition: In T, $\exists\ e_c\ (n,\ x_c,\ d)$, $e_p\ (n_p,\ x_p,\ n) \in E \wedge d \in D \wedge n \in N$
Postcondition: In T', $E' = E \setminus \{e_c\ (n,\ x_c,\ d),\ e_p\ (n_p,\ x_p,\ n)\} \wedge N' = N \setminus \{n\} \wedge D' = D \wedge X' = X \setminus \{x_p, x_c\}$

For convenience, we define three additional non-primitive operators that can be derived from the four perturbation primitive operators. Two insert a subtree and delete a subtree. One changes the edge between a node and its data type.

1) $insertT\ (n,\ e',\ T^a)$
Action: Insert a new tree T^a below node n. A new edge e' connects n to the root node of T^a.
Precondition: For T, $n \in N \wedge T^a = (N^a,\ D^a,\ X^a,\ E^a,\ n^a_r)$
Postcondition: In T', E' = $E \cup \{e'\ (n,\ x',\ n^a_r)\} \cup E^a \wedge N'$=$N \cup N^a \wedge X'$=$X \cup \{x'\} \cup X^a \wedge D'$=$D \cup D^a$

2) $deleteT\ (e)$
Action: Delete a subtree from the tree, starting from the edge e. The parent node of e must have at least one other child node.
Precondition: For T, $\exists\ e\ (n_p,\ x,\ n_c)$, $e_2\ (n_p,\ x,\ n_{c2}) \in E \wedge n_c,\ n_{c2} \in N$
Postcondition: In T', $\exists\ T^d\ (N^d,\ D^d,\ X^d,\ E^d,\ n^d_r) \subset T \wedge n^d_r$ = $n_c \wedge N' = N \setminus N^d \wedge X' = X \setminus X^d \wedge D' = D \wedge E' = E \setminus (E^d \cup \{e\})$

3) $changeE\ (e, e')$
Action: Change the edge e to e'. e' is still between the same pair of nodes, but the constraints on e are perturbed. The allowable constraints for elements, attributes, and each data type are as defined in XML Schema Recommendation 2.
Precondition: In T, $e\ (n_p,\ x,\ n_c) \in E$

Postcondition: In T', $E' = (E \setminus \{e\ (n_p,\ x,\ n_c)\}) \cup \{e'\ (n_p,\ x',\ n_c)\} \wedge N' = N \wedge D' = D \wedge X' = (X \setminus \{x\}) \cup \{x'\}$

These six operators are used to develop test criteria in Section 5.

5. Test Coverage Criteria

This section defines test coverage criteria based on the XML data model and perturbation operators. The criteria use the structure and constraints defined by the schemas, and model specific mistakes that can occur in the XML in terms of the perturbation operators in the previous section. Two criteria, delete coverage and insert coverage, modify the structure of an XML schema. Another criterion, constraint coverage, uses the changeE() operator to modify the constraint values.

Two operations are involved in perturbation operators: insert ($insertND()$ and $insertN()$) and delete ($deleteN()$ and $deleteND()$). The intent is to mimic likely user errors by creating XML schemas that differ only slightly from the original XML schemas. Delete coverage makes all possible delete operations on a tree by using the two delete perturbation operators. All nodes that do not define a data type are modified by the primitive $deleteN()$. All nodes that are defined by a data type are modified by the primitive $deleteND()$. Formally, the delete coverage criterion is:

> **Delete Coverage (DC)**: Given a tree $T = (N,\ D,\ X,\ E,\ n_r)$, the set of test requirements (TR) includes two sets of operations: $\{deleteN\ (n_1), ..., deleteN\ (n_j)\}$, where $n_1, ..., n_j$ is the list of nodes that satisfy $\forall\ e_p\ (n_p,\ x_p,\ n)$, $e_c\ (n,\ x_c,\ n_c) \in E \wedge n_p,\ n_c,\ n \in N$, and $\{deleteND\ (n_1), ..., deleteND\ (n_j)\}$, where $n_1, ..., n_j$ is the list of nodes that satisfy $\exists\ e_c\ (n,\ x_c,\ d),\ e_p\ (n_p,\ x_p,\ n) \in E \wedge d \in D \wedge n \in N$.

Insert coverage is based on delete coverage by adding insertions after deletions. For each $deleteN()$ operation applied during delete coverage, $insertN()$ operates on all edges that connect two nodes, using the deleted node as input. For each $deleteND()$ operation applied during delete coverage, $insertND()$ operates on all nodes that do not define a data type, using the deleted node and data type as the inputs. Formally, the insert coverage criterion is:

> **Insert Coverage (IC)**: Given a tree $T = (N,\ D,\ X,\ E,\ n_r)$, $\{T' = (N',\ D',\ X',\ E',\ n_r)\}$ is the set of trees after delete operations $\{\sum deleteN(n), \sum deleteND(n)\}$ are applied during delete coverage, where for $deleteN()$, n must satisfy $\forall\ e_p\ (n_p,\ x_p,\ n)$, $e_c\ (n,\ x_c,\ n_c) \in E \wedge n_p, n_c, n \in N$ and for $deleteND()$, n must satisfy $\exists\ e_c\ (n,\ x_c,\ d)$, $e_p\ (n_p,\ x_p,\ n) \in E \wedge d \in D \wedge n \in N$.
>
> The set of test requirements (TR) includes two sets of operations: $\{\sum insertN\ (e,\ e'_p,\ e'_c,\ n)\}$, where $e \in E'$, $e'_p = (n'_p,\ x_p,\ n)$, $e_c = (n,\ x_c,\ n'_c)$, and n'_p and n'_c must satisfy the precondition of the $insertN()$ operation on T'. Also, TR includes another set $\{\sum insertND\ (n_p,\ e'_p,\ n',\ e'_c,\ d')\}$, where $n_p \in N'$, $e_c = (n,\ x_c,\ d)$, $e_p = (n_p,\ x_p,\ n)$, $d' = d$, and n_p must satisfy the precondition of the $insertND()$ operation on T'.

Constraints implement business rules in XML schema. In Section 3, for example, the business rule that the maximum price of a book is $1000 is realized as a representation constraint in the XML schema for books. The constraint coverage criterion covers all constraints in the X set. It uses the perturbation operator $changeE()$ to modify the constraint values. The facets in constraints are not modified. Formally, the constraint coverage criterion is:

> **Constraint Coverage (CC)**: Given a tree $T = (N,\ D,\ X,\ E,\ n_r)$, the set of test requirements (TR) includes one set of operations: $\{changeE\ (e_i,\ e'_i), ..., changeE\ (e_j,\ e'_j)\}$, where $e_i, ...,\ e_j$ are all the edges that satisfy $e : (n_p,\ x, n_c) \in E \wedge x \neq \phi$.

The three coverage criteria are illustrated with the XML schema for books. The delete operation set is: $\{deleteN(book),\ deleteND(ISBN),\ deleteND(price),\ deleteND(year)\}$. The trees created by applying the delete operations are shown in Figure 2. Figure 2.a is the tree after the deleteN (book) operation is applied, and Figure 2.b is the tree after deleteND (year) is applied. Two other delete operations, deleteND (ISBN) and deleteND (price), are similar to deleteND (year). For operation deleteN (book), the insert operation set is $\{insertN(e_2, e_1, e_2, book),\ insertN(e_3, e_1, e_3, book),\ insertN(e_4, e_1, e_4, book)\}$. For operation deleteND (year), the insert operation set is $\{insertND(\ book,\ e_4, year, e_7, int),\ insertND(books,\ e_4,\ year,\ e_7, int)\}$. But the $insertND(book, e_4, year, e_7, int)$ results in the original XML schema, so is not used. Two other delete operations, deleteND (ISBN) and deleteND (price), are similar to the deleteND (year). The XML schema for the books has two constraints in the X set. The constraint coverage is the operation set: $\{changeE(e_1, e'_1),\ changeE(e_6, e'_6),\ changeE(e_6, e''_6)\}$.

The number of operations defined on an XML schema is bounded as follows. In delete coverage, given a tree $T = (N,\ D,\ X,\ E,\ n_r)$, there exist two sets $N_n \subset N$ and $N_d \subset N$, where for each $n \in N_n$, there must exist $e_c(n, x_c, n_c) \in E \wedge n_c \in N$, and for each $n \in N_d$, there must exist $e_c(n, x_c, d) \in E \wedge d \in D$. So, the number of operations in delete coverage is $(|N_n| - 1) + |N_d|$.

Insert coverage is based on delete coverage, so the number of insert operations based on the deleteN operator is

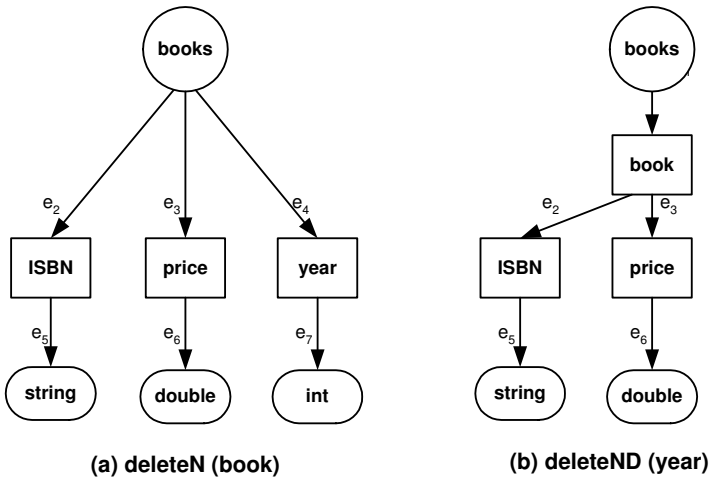

Figure 2. The perturbed trees as a result of the delete operations.

$(|N| - 2) \times (|N_n| - 1)$. The number of insert operation based on the deleteND operator is $|N_n| \times |N_d|$. So, the total number of insert operations is $(|N| - 2) \times (|N_n| - 1) + |N_n| \times |N_d|$.

Finally, the number of operations in constraint coverage is $|X|$.

Consider the book schema example. It contains two nodes that do not have data types, books and book, so $|N_n| = 2$. Three nodes are defined with data types, ISBN, price, and year, so $|N_d| = 3$. The number of XML schemas created in delete coverage is $(2 - 1) + 3 = 4$. The number of XML schemas created in insert coverage is $(5-2) \times (2-1) + 2 \times 3 = 9$. The number of XML schemas created in constraint coverage is 3, for a total of 16.

6. Generating Test Cases

When validating software components that communicate via XML messages, a test case is an XML message. The approach in this paper generates tests from perturbed XML schemas. The most difficult part of generating the tests is finding values to satisfy the constraints. Constraints are defined between pairs of nodes and between nodes and data types. Constraints between nodes describe integrity relationships between nodes and so are called *integrity constraints*. XML Schema Recommendation 2 identifies five types of constraints, "unique," "maxOccurs," "minOccurs," "nillable," and "use." Constraints between nodes and data types express formats that the data values must conform to, so are called *representation constraints*. In fact, these representation constraints are the facet constraints defined in XML Schema Recommendation 2. Both types of constraints are classified as *boundary constraints* and *non-boundary constraints*. Table 1 lists the boundary and non-boundary constraints.

Boundary Constraints	**Non-boundary Constraints**
maxOccurs, minOccurs, length, maxExclusive, maxInclusive, maxLength, minExclusive, minInclusive, minLength, totalDigits	enumeration, use fractionDigits, pattern, nillable, whiteSpace, unique

Table 1. Classification of the constraints.

Different strategies are used to generate tests from boundary and non-boundary constraints. For each boundary constraint, a *boundary value* is defined and generated. For example, the books schema has the boundary constraint "maxInclusive = 1000.00" for the element **price**. The boundary value is the maximum number allowed for that constraint, 1000. For each non-boundary constraint, an arbitrary (random) value that conforms to the constraint is generated. Consider the constraint "fractionDigits = 2". The test case might be 349.99. If one edge has two constraints, two separate tests are generated. If an edge has no constraint, an arbitrary value is generated. There are two kinds of edges without constraints, an edge between a node and a data type, and an edge between two nodes. For an edge between a node and a data type, the arbitrary value must conform to the data type. For an edge between a node and a data type, the arbitrary value is just one child node instance generated.

Different combinations of data in an XML message can yield different results. Therefore, multiple XML messages

need to be generated for the same XML schema. Our strategy focuses on the constraints. One XML message is generated for each constraint based on whether this constraint is a boundary constraint, and arbitrary values are generated for other constraints. Therefore, given a modified tree $T = (N, D, X, E, n_r)$, there are $|X|$ test cases if some constraints exist, otherwise there is only one test case.

Although the goal of this process is to generate invalid XML messages, some valid messages can be generated. Valid XML messages are not considered useful tests, and can introduce "false positives." That is, if a valid XML message is created and the software produces a valid response, the tester may believe that a failure occurred in the software. These tests are currently identified by hand, but we hope to find ways to automatically identify valid XML messages before executing them.

Consider the books schema. The constraint set X has three constraints on two different edges, so three tests are created. These tests are shown as simplified XML messages in Example 2.

Example 2 *: Books Test Cases as Simplified XML Messages*

```
<books>
   <!- - test case 1 for maxOccurs - ->
      <book>
         <ISBN>0-201-74095-8<ISBN/>
         <price>37.50<price/>
         <year>2002<year/>
      </book>
   <!- - omit other elements - ->
</books>

<books>
   <!- - test case 2 for maxInclusive - ->
      <book>
         <ISBN>0-201-74095-8<ISBN/>
         <price>1000.00<price/>
         <year>2002<year/>
      </book>
</books>

<books>
   <!- - test case 3 for fractionDigits - ->
      <book>
         <ISBN>0-201-74095-8<ISBN/>
         <price>349.99<price/>
         <year>2002<year/>
      </book>
   </books>
```

7. Empirical Evaluation I: MARS

This paper contains results from two case studies. Results from a small web service, the Mars Robot Communication System (MARS), are presented in this section. This small application was developed by the first author, who also carried out the testing process. To avoid bias, the second empirical study used an an open source web service application. Results from this application are in the following section. We used the TCPMonitor in the Apache Axis [25] to monitor the messages transferred between web services, manually generated the perturbed XML schema and XML messages, and resent the new messages through the TCPMonitor. Except for the use of TCPMonitor, the testing was done by hand.

As was discussed at the recent Workshop on Testing, Analysis and Verification of Web Services [21], it is very difficult to find web services that researchers can experiment with. Many of those that exist are very small and use little communication [16]. Thus, our study uses a collection of web services developed specifically for this research. The Mars Robot Communication System (MRCS) supports a scientific mission to Mars by carrying geological and meteorological data from Mars to a space station in orbit around Earth, then to a ground control system that stores the data in a database and provides access to scientists. MRCS has three separate major components: A robot on Mars, the space station, and ground control in Houston. The ground control system is a 3-tier web application that includes the web server, an application server, and a database server. All communications between the three components and among the servers in the ground control system are in XML under SOAP.

A SOAP envelope contains a SOAP body and a header. The body can contain a fault section. If a web service *A* sends a request to another service *B*, and *B* cannot process the data in the request, *B* sends a message back with a fault section to tell *A* that its request was invalid.

We define an *abnormal response* to occur in two situations. First, if *B* cannot process the data and returns a message with a fault section, that is an abnormal response. Second, if *B* has a runtime exception and returns no message, that is an abnormal response. A *normal* response is when *B* returns an XML message wrapped in SOAP with no fault section. Our previous paper [20] presented valid tests; the tests developed in this research are invalid tests, thus the correct response from the web services should be **abnormal** responses on all tests.

Four schemas were defined for MRCS, so our study did not need to create virtual schema. The test results are shown in Table 2. 63 perturbed schemas were generated, and resulted in 232 invalid XML messages (valid XML messages were eliminated by hand during creation). The 42 normal responses in Table 2 represent errors that the tests found in the web service behavior.

	DC	IC	CC	Original	Total
XML Schemas	23	25	11	4	63
XML Messages	64	103	53	12	232
Abnormal response	64	125	11	0	190
Normal response	0	0	42	12	42

Table 2. The results from test case generation.

Schema	$\#N_n$	$\#N_d$	# X	DC	IC	CC
retailer	4	11	9	14	83	9
warehouse	3	4	3	6	22	3
manufacturer	3	6	3	8	32	3

Table 3. Information about the formal model of the schemas.

Schema	DC	IC	CC
retailer	129	710	90
warehouse	15	48	6
manufacturer	14	48	3

Table 4. The number of XML messages for the three coverage criteria.

8. Empirical Evaluation II: SCMSA

The Supply Chain Management Sample Application (SCMSA) is a sample application defined by the Web Services Interoperability Organization (WS-I) [1, 9]. WS-I's goal is to help developers create and deploy interoperable web services. SCMSA models a retailer that offers consumer electronic goods to consumers. This is a typical B2C model. To fulfill orders, the retailer has to manage stock levels in warehouses (warehouseA, warehouseB, and warehouseC). When an item in stock falls below a certain threshold, the retailer must restock items from the relevant manufacturers' inventory (manufacturerA, manufacturerB, manufacturerC). Retailer, Logging, warehouseA, warehouseB, warehouseC, manufacturerA, manufacturerB, and manufacturerC are published as eight separate web services. The application use cases defined in the Supply Chain Management Use Case Model [1]. Technical design and implementation details are documented in the Supply Chain Management Architecture document [9]. These specifications and design were implemented by the first author.

The XML schema for SCMSA were analyzed, faults were seeded by hand into the implementation, and tests were generated and run against the faults.

8.1. SCMSA Schema Analysis

SCMSA defines seven XML schema.

1. `configuration.xsd`: This schema defines the types needed to describe the header common to all services.
2. `manufacturerSN.xsd`: Shipment Notification schema for manufacturer system.
3. `manufacturerPO.xsd`: Purchase Order schema for manufacturer system.
4. `retailerOrder.xsd`: Definition of PartsOrder types for Retailer component.
5. `retailerCatalog.xsd`: Catalog schema for retailer component.
6. `warehouse.xsd`: The schema for the warehouse component.
7. `catalog.xsd`: Schema for Catalog web service.

Since we only need to perturb the request messages, we only consider the three request XML schema `retailer-req.xsd`, `manufacturer-req.xsd`, and `warehouse-req.xsd`, which are derived from the original seven schemas. The three criteria delete coverage (DC), insert coverage (IC), and Constraint Coverage (CC) were applied to the three XML schema. Table 3 summarizes some information about these schema. In the table, $\#N_n$ is the number of the nodes that have child nodes, $\#N_d$ is the number of the nodes that have no child nodes and has an associated data type, and #X is the number of constraints. The columns DC, IC and CC give the number of XML schemas generated by each coverage criterion.

The number of XML messages (that is, tests) depends on the number of constraints. We only generate XML messages for boundary constraints. The number of XML messages generated for each of the three coverage criteria are listed in Table 4. The schema analysis and test creation was by the first author. Table 4 only shows invalid XML messages; valid XML messages were eliminated by hand during creation.

8.2. Seeded Faults

Our testing strategy is designed to test server-side components. Web services are often "peer-to-peer," so there is no default server-side. This study makes the simplifying assumption that certain components are server-side, in particular, the retailer, warehouse, and manufacturer components.

In reality, the retailer might be the client of the warehouse, but this part of their organization is not critical to this study. Twenty-one faults were inserted by hand by the third author and distributed among the five Java classes. The faults were not derived from a fault model (none seems to exist for web services), so they were based on intuition. We did not insert faults into the back-end beans or database connection program.

8.3. Experimental Results

Our tests detected seven of the 21 faults. All seven faults were detected by the constraint coverage (CC) tests, and none by the DC or IC tests. A detailed analysis of the faults revealed four reasons why some faults were not found by the tests.

1. Six faults were such that they either could not be found by external inputs (that is, XML), or could not be found in the current configuration of the web service. Most were related to initializations of objects and the scope of object declarations. Thus, these faults could only be found during unit testing and are out of scope of the techniques in this paper.

2. Five faults only affected the back-end of the web service, not the service response. They caused erroneous information to be written to a log file. At least one test caused each of these faults to result in a failure, however, our test process is such that only the web service response was evaluated, thus we had no way to detect these failures and they are considered to be not found. This is a common issue with testing web applications and web services and is discussed below.

3. One fault depends on inputs that are defined in the backend database. Our tests only include the XML messages, so this fault is out of scope of this type of testing. Another test strategy is needed to find this type of fault.

4. Two faults could have been found, but causing them to result in a failure required using specific input values that we did not happen to create. We are currently investigating ways to detect more of these faults.

All tests generated for DC and IC were refused at the beginning of the processing. That is, the software was robust enough to recognize them as being invalid. It should not be surprising that a well built web service would catch this kind of relatively simple mistake. It remains to be seen whether other web services will do so well. However, the software did not do as well in conforming to the semantic constraints. Again, this is not surprising, because these are harder to analyze.

9. Conclusions and Future Work

This paper has introduced a novel test method for XML-based communication. A formal model for XML schema is defined and a method to create virtual schema when XML schemas do not exist is presented. Based on this formal model, perturbation operators are designed. The operators are used to modify virtual or real schema to satisfy defined test coverage criteria. Test cases, as XML messages, are generated to satisfy the perturbed XML schema. The method was applied as a case study to two web services.

As case studies, there is no expectation of external validity of the results. That will require further experimentation. A threat to internal validity is that the first author performed all steps in generating and executing tests. An additional scientist was used in the project specifically to seed faults into the SCMSA web service to avoid the bias of having the same person generate tests and create faults.

Our most important future work on this project is to automate the generation and execution of tests. We are actively working on a tool to do this.

The empirical results suggest several avenues for future research. One problem with testing web applications and web services is that of *observability*, as defined by Binder [6]. Specifically, when web services change files and databases on the server, it is very difficult to observe incorrect outputs. With web services, we often do not have direct access to the data stores on the server. As of now, we are not aware of any proposed solutions to this problem.

A related problem is that of *controllability* [6]. When web services use data from the backend server, these values cannot be supplied (or controlled) by external testers. Thus, evaluating behavior with regard to those inputs is a responsibility of the developers during unit and module testing.

The results also bring up questions about the perturbation operators designed for this research. Delete Coverage (DC) and Insert Coverage (IC) were ineffective. We cannot yet be sure whether this is because they are inherently not useful or if the software we evaluated was particularly good at screening invalid inputs. We are continuing to evaluate these perturbation operators, as well as design new perturbation operators that can detect additional faults.

References

[1] S. Anderson, M. Chapman, M. Goodner, P. Mackinaw, and R. Rekasius. Supply chain management use case model. online WS-I specification document, 2003. http://www.ws-i.org/SampleApplications/SupplyChainManagement/2003-04/SCMUseCases1.0-BdAD.pdf, last access August 2005.

[2] A. Andrews, J. Offutt, and R. Alexander. Testing Web applications by modeling with FSMs. *Software and Systems Modeling*, 4(3), August 2005.

[3] M. Aoyama, S. Weerawarana, H. Maruyama, C. Szyperski, K. Sullivan, and D. Lea. Web services engineering: Promises and challenges. In *Proceedings of the 24th International Conference on Software Engineering*, pages 647–648, Orlando, Florida, May 2002.

[4] M. Arenas, W. Fan, and L. Libkin. What's hard about XML Schema constraints? In *The International Conference on Database and Expert Systems Applications (DEXA)*, pages 269–279, Springer, Heidelberg, 2002.

[5] M. Benedikt, J. Freire, and P. Godefroid. Veriweb: Automatically testing dynamic Web sites. In *Proceedings of 11th International World Wide Web Conference (WW W'2002)*, Honolulu, HI, May 2002.

[6] B. Binder. *Testing Object-oriented Systems*. Addison-Wesley Publishing Company Inc., New York, New York, 2000.

[7] J. Bloomberg. Testing Web services today and tomorrow. The Rational Edge E-zine for the Rational Community, October 2002. http://www-128.ibm.com/developerworks/rational/library/content/Rational% -Edge/oct02/WebTesting_TheRationalEdge_Oct02.pdf, last access August 2005.

[8] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: A new approach to design and analysis of e-service composition. In *The Twelfth International World Wide Web Conference (WWW2003)*, Budapest, Hungary, May 20-24 2003.

[9] M. Chapman, M. Goodner, B. Lund, B. Mckee, and R. Rekasius. Supply chain management sample application architecture. online WS-I specification document, 2003. http://www.ws-i.org/SampleApplications/SupplyChainManagement/2003-04/SCMArchitecture1.0-BdAD.pdf, last access August 2005.

[10] J. Clabby. *Web services explained: Solutions and applications for the real world*. Pearson Education Inc., 2003.

[11] N. Davidson. Web services testing. The Redgate software technical papers, 2002. http://www.redgate.com/dotnet/more/web_services_testing.htm, last access August 2005.

[12] S. Elbaum, S. Karre, and G. Rothermel. Improving Web application testing with user session data. In *Proceedings of the 25th International Conference on Software Engineering*, pages 49–59, Portland, Oregon, May 2003. IEEE Computer Society Press.

[13] W. Fan, G. Kuper, and J. Simeon. A unified constraint model for XML. In *10th International World Wide Web Conference (WWW10)*, pages 179–190, Hong Kong, China, May 2001.

[14] X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL Web services. In *The Twelfth International World Wide Web Conference (WWW2004)*, New York, New York, May 17-22 2004.

[15] M. R. A. Huth and M. D. Ryan. *Logic in Computer Science*. Cambridge University Press, 2000.

[16] IBM. IBM web services. online, 2000. http://alphaworks.ibm.com/webservices, last access August 2005.

[17] IBM. Web services: Taking e-business to the next level. IBM white paper, 2000. http://www-900.ibm.com/developerWorks/cn/wsdd/download/pdf /e-businessj.pdf, last access August 2005.

[18] C. Kirkegaard, A. Moller, and M. I. Schwartzbach. Static analysis of XML transformations in Java. *IEEE Transactions on Software Engineering*, 30(3):181–192, March 2004.

[19] J. Offutt, Y. Wu, X. Du, and H. Huang. Bypass testing of web applications. In *15th International Symposium on Software Reliability Engineering*, pages 187–197, Saint-Malo, Bretagne, France, November 2004. IEEE Computer Society Press.

[20] J. Offutt and W. Xu. Generating test cases for Web servcies using data perturbation. In *Workshop on Testing, Analysis and Verification of Web Services*, pages 41–50, Boston, Massachusetts, July 2004.

[21] Organizers: T. Bultan and S. Krishnamurthi. Workshop on testing, analysis and verification of web services (TAVWEB 2004). Online, July 2004. http://www.cs.ucsb.edu/ bultan/tav-web/, last access August 2004.

[22] F. Reuter and N. Luttenberger. Cardinality constraint automata: A core technology for efficient XML schema-aware parsers. In *The Twelfth International World Wide Web Conference (WWW2003)*, Budapest, Hungary, May 2003.

[23] F. Ricca and P. Tonella. Analysis and testing of web applications. In *23rd International Conference on Software Engineering (ICSE '01)*, pages 25–34, Toronto, CA, May 2001.

[24] H. Su, D. Kramer, L. Chen, K. Claypool, and E. A. Rundensteiner. Xem: Managing the evolution of XML document. In *Research Issues in Data Engineering, 2001. Proceedings. Eleventh International Workshop on*, pages 103–110, April 2001.

[25] The Axis Development Team. The Apache web services - Axis web page, 2005. http://ws.apache.org/axis/, last access August 2005.

[26] W3C. Extensible markup language (XML) 1.0 (second edition) – W3C recommendation, October 2000. http://www.w3.org/XML/.

[27] W3C. XML schema – recommendation, May 2001. http://www.w3c.org/tr/.

[28] W3C. XML schema part 1: Datatypes. W3C Recommendation, 2001. http://www.w3.org/TR/xmlschema-2/, last access April 2004.

[29] W3C. XML schema part 2: Structures. W3C Recommendation, 2001. http://www.w3.org/TR/xmlschema-1/, last access April 2004.

[30] J. Williams. The Web services debate: J2EE vs. .NET. *E-services: A cornucopia of digital offerings ushers in the next Net-based evolution*, 46(6):58–63, June 2003.