Grammarware Application:
Testing XML Validators

Vadim Zaytsev

26 November 2004
The story of one grammar-based tool
Grammarware and XML

- As it was told, grammarware is more than just compilers!
- eXtensible Markup Language — has a grammar (XML Schema)
- XML validator is a grammar-based tool:
Grammarware and XML
XML Schema is also a language

- And as such, it has a grammar
- Generate concrete grammars from the grammars’ grammar
- Official name: XML Schema Schema for XML Schemas
XML Schema is also a language

XML

Validator

Test Data Generator

XSD

Oracle

Y/N

GOOD/BAD

YES

NO
Differential testing

- Why Oracle?

- Having several XML validators, we can set them up to play against one another:
  - A file is fed to all of them
  - Diagnoses are gathered
  - If all agreed, cool
  - Different outputs reveal bugs
Differential testing
Combinatorial testing

• How to choose what to test?
• Let the grammar decide! Produce everything possible!
• Complementary to stochastic testing
• Characteristics:
  • No randomisation; no heuristics
  • Detailed control mechanisms
  • Formally defined coverage
  • Focus on huge test-data sets
  • Addresses grammar-based software
Combinatorial testing
Combinatorial testing

Grammar

Term

Term

Term

...
Explosion

- Why not feasible?
  - Number of terms grows fast with depth
  - Grammars are complex

- *Explosion* means exponential behaviour

- Number of terms gets unfeasible within a very small number of depth layers explored
Number of generated terms grows fast with depth and eventually explodes (becomes greater than 18446744073709551616).
Solution? *Controlled* explosion

- Explosion is going to happen.

- We can try to postpone (to control) it.

- Now a tester’s intuition comes into play.

- (in a strictly formalised way, though)
Controlled explosion

Grammar → Recursion control → Depth control

Term → Term → Term → Term → Term

... other mechanisms
Control mechanisms*

- **Depth control** — “length” of terms
- **Recursion control** — nested constructor applications
- **Equivalence control** — build equivalence classes
- **Balance control** — limit preceding levels
- **Combination control** — limited arguments use
- **Context control** — enforce context conditions

Depth control

Taken from XHTML Strict 1.0 XML Schema:

```xml
<xs:group name="head.misc">
  <xs:sequence>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="script"/>
      <xs:element ref="style"/>
      <xs:element ref="meta"/>
      <xs:element ref="link"/>
      <xs:element ref="object"/>
    </xs:choice>
  </xs:sequence>
</xs:group>
```

Nobody is interested in infinite `<head>` tag.
Recursion control

Adopted from XHTML Strict 1.0 XML Schema:

```xml
<x:s:element name="span">
  <xs:complexType mixed="true">
    <xs:complexContent mixed="true">
      <xs:extension base="Inline">
        <xs:attributeGroup ref="attrs"/>
      </xs:extension>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
...
<x:s:complexType name="Inline" mixed="true">
  <xs:choice minOccurs="0" maxOccurs="unbounded">
    <xs:element ref="span"/>
  </xs:choice>
</xs:complexType>
```

We prefer to go deeper without a burden of nested `<span>`s.
Combination control

Taken from XHTML Strict 1.0 XML Schema:

```xml
<xs:attributeGroup name="events">
    <xs:attribute name="onclick" type="Script"/>
    <xs:attribute name="ondblclick" type="Script"/>
    <xs:attribute name="onmousedown" type="Script"/>
    <xs:attribute name="onmouseup" type="Script"/>
    <xs:attribute name="onmouseover" type="Script"/>
    <xs:attribute name="onmousemove" type="Script"/>
    <xs:attribute name="onmouseout" type="Script"/>
    <xs:attribute name="onkeypress" type="Script"/>
    <xs:attribute name="onkeydown" type="Script"/>
    <xs:attribute name="onkeyup" type="Script"/>
</xs:attributeGroup>
```

XML attributes are numerous, but often independent.
Some XML validators

- .NET API — C#-based validator
  - simple wrapper had to be written
- JAXB — Sun Multi-Schema XML Validator 1.2
  - Java-based, free of charge
- Python — XSV
  - [http://www.w3.org/2001/03/webdata/xsv](http://www.w3.org/2001/03/webdata/xsv)
  - free of charge, used by the W3C
  - simple wrapper had to be written
Some XML validators

C#

Java

Python
Scalability issues

- Opening the directory
  - Windows Explorer does not work
  - light-weight file managers give up at 1M
- Copying files
  - takes hours to complete
- FOR in Windows (.bat file syntax)
  - does not work with more than 15k files
  - silently skips $\approx 0.03\%$ of the files
- “*” in Linux
  - core dumped
- Editing files
  - XML Spy gives in on too complicated files
  - Visual Studio .NET 2003 works!
Scalability issue
Scalability issue
What to test in the XML?

- Levels of XML file conformance

- Levels of XML processor conformance

- Grammar features: attributes, references, . . .

- Advanced features: namespaces, schema-related markup, . . .

- Secondary features: header, scalability, . . .
Before validity comes...

- Well-formedness
  - the document as a whole matches the production document
  - all tags closed in place

- Proper header:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html
    PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
     xml:lang="en" lang="en">
</html>
```
Attributes and “simple” types

Taken from XHTML Strict 1.0 XML Schema:

```xml
<xs:simpleType name="Length">
  <xs:restriction base="xs:string">
    <xs:pattern value="[-+]?(\d+|\d+(\.\d+)?)%"/>
  </xs:restriction>
</xs:simpleType>

<xs:simpleType name="MultiLength">
  <xs:restriction base="xs:string">
    <xs:pattern value="[-+]?(\d+|\d+(\.\d+)?)|[1-9]?\d+\*"/>
  </xs:restriction>
</xs:simpleType>

<xs:element name="img">
  <xs:complexType>
    <xs:attribute name="height" type="Length"/>
    <xs:attribute name="width" type="Length"/>
    ...
  </xs:complexType>
</xs:element>
```

One of the problems found: duplicate attributes!
Document-wide unique identifiers

Taken from XHTML Strict 1.0 XML Schema:

```xml
<x:s:element name="html">
  <x:s:complexType>
    ...
    <x:s:attribute name="id" type="xs:ID"/>
  </x:s:complexType>
</x:s:element>
...
<x:s:element name="td">
  <x:s:complexType mixed="true">
    <x:s:complexContent mixed="true">
      <x:s:extension base="Flow">
        <x:s:attribute name="headers" type="xs:IDREFS"/>
      </x:s:extension>
      ...
    </x:s:complexContent>
  </x:s:complexType>
</x:s:element>
```
Namespaces

Taken from Namespaces in XML:

```xml
<?xml version="1.0"?>
<!-- initially, the default namespace is "books" -->
<book xmlns='urn:loc.gov:books'
  <title>Cheaper by the Dozen</title>
  <isbn:number>1568491379</isbn:number>
  <notes>
    <!-- make HTML the default namespace for some commentary -->
    <p xmlns='urn:w3-org-ns:HTML'>
      This is a <i>funny</i> book!
    </p>
  </notes>
</book>
```

Different document parts may belong to different namespaces and conform to different XML Schemas.
Validator’s tolerance

- *Lax* validation in the XSV
  - activated automatically with an empty schema

- Unknown element
  - .NET warning

- Validator’s robustness
  - XSV crashes with a duplicate attribute
  - stress testing (stress nesting)
How does it work

- XSD file is parsed
- additional grammar file is parsed
- their contents form a grammar
- terms are generated in memory
- terms are serialised as XML files to the hard disk
How does it work
Visualisation

• after parsing is over the complete grammar is dumped

• during generation we can see number of terms per sort

• generation process can be paused

• we can stop at any depth
Visualisation
Visualisation
Conclusion

- XML validator tests an XML file to conform to a grammar
- XML Schema is not an easy spec to implement (to test)
- Our tool tests if an XML validator works well
- \textit{Automated} generation of huge test-data sets
- \textit{Differential} testing for race of validators
- \texttt{http://www.cs.vu.nl/grammarware}
Questions?
The hierarchy of XML files processing

XML Validator
- XML Validation API
- XML API
- Framework
- Hardware Platform