COMP60411: Modelling Data on the Web

SAX, Schematron, JSON, Robustness & Errors

Week 4

Bijan Parsia & Uli Sattler
University of Manchester

SE2 General Feedback

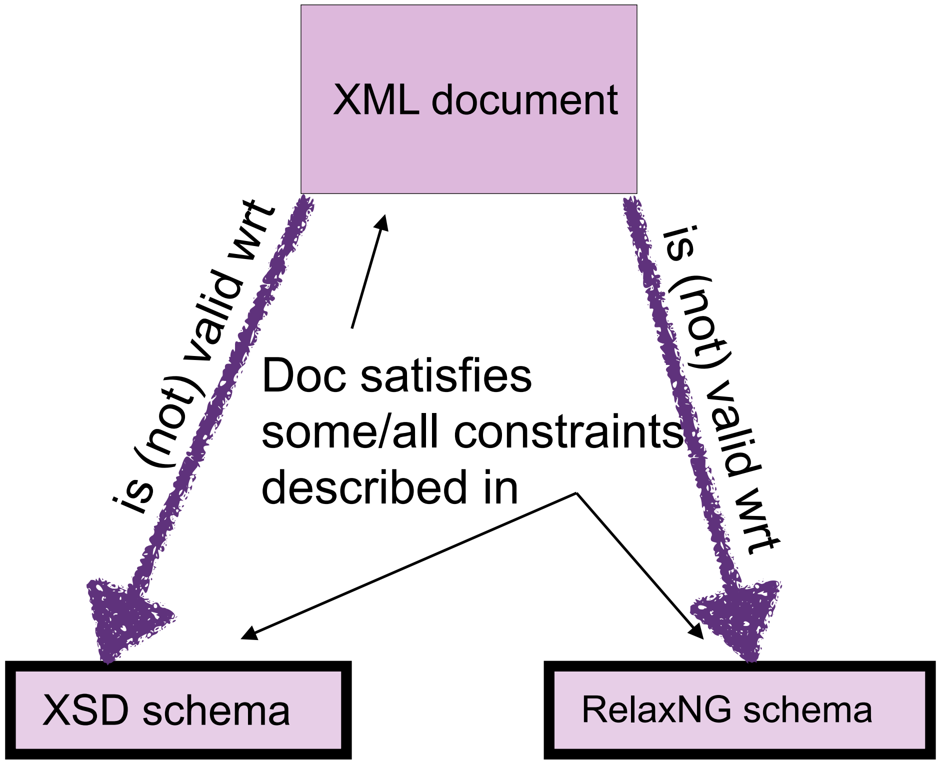
- use a good spell checker
- answer the question
  - ask if you don’t understand it
  - TAs in labs 15:00-16:00 Mondays - Thursdays
  - we are there on a regular basis
- many confused “being valid” with “validate”

[…] a situation that does not require input documents to be valid (against a DTD or a RelaxNG schema, etc.) but instead merely well-formed.

- read the feedback carefully
  - including the one in the **rubric**

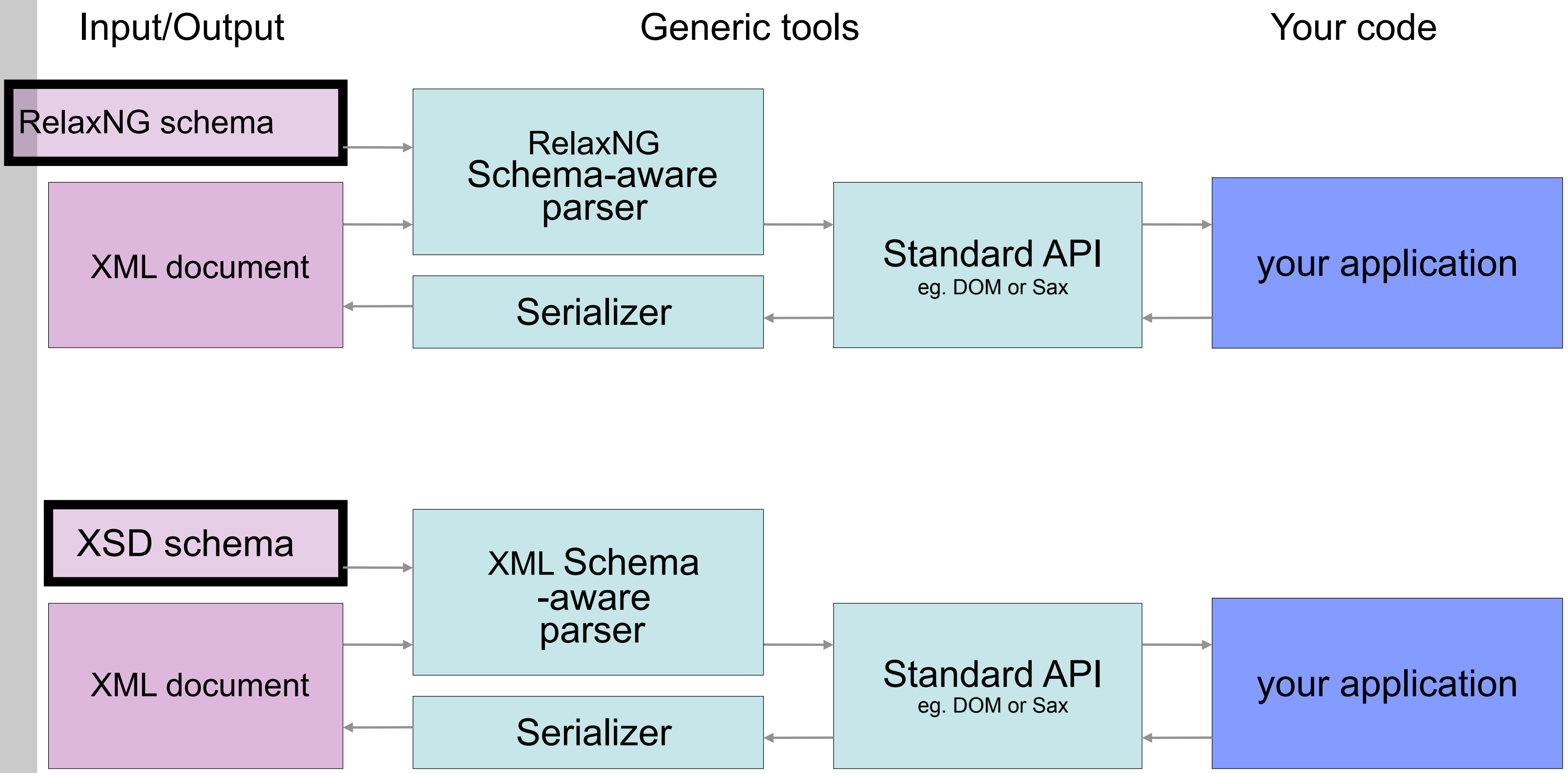
Being valid wrt a schema in some schema language

One even called **XML Schema**



Validating a document against a schema

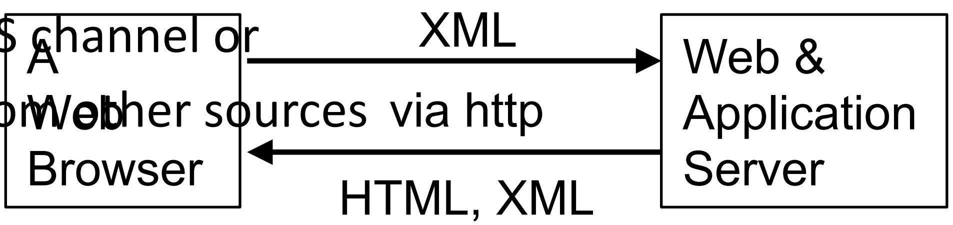
in some schema language



SE2 General Feedback: applications using XML

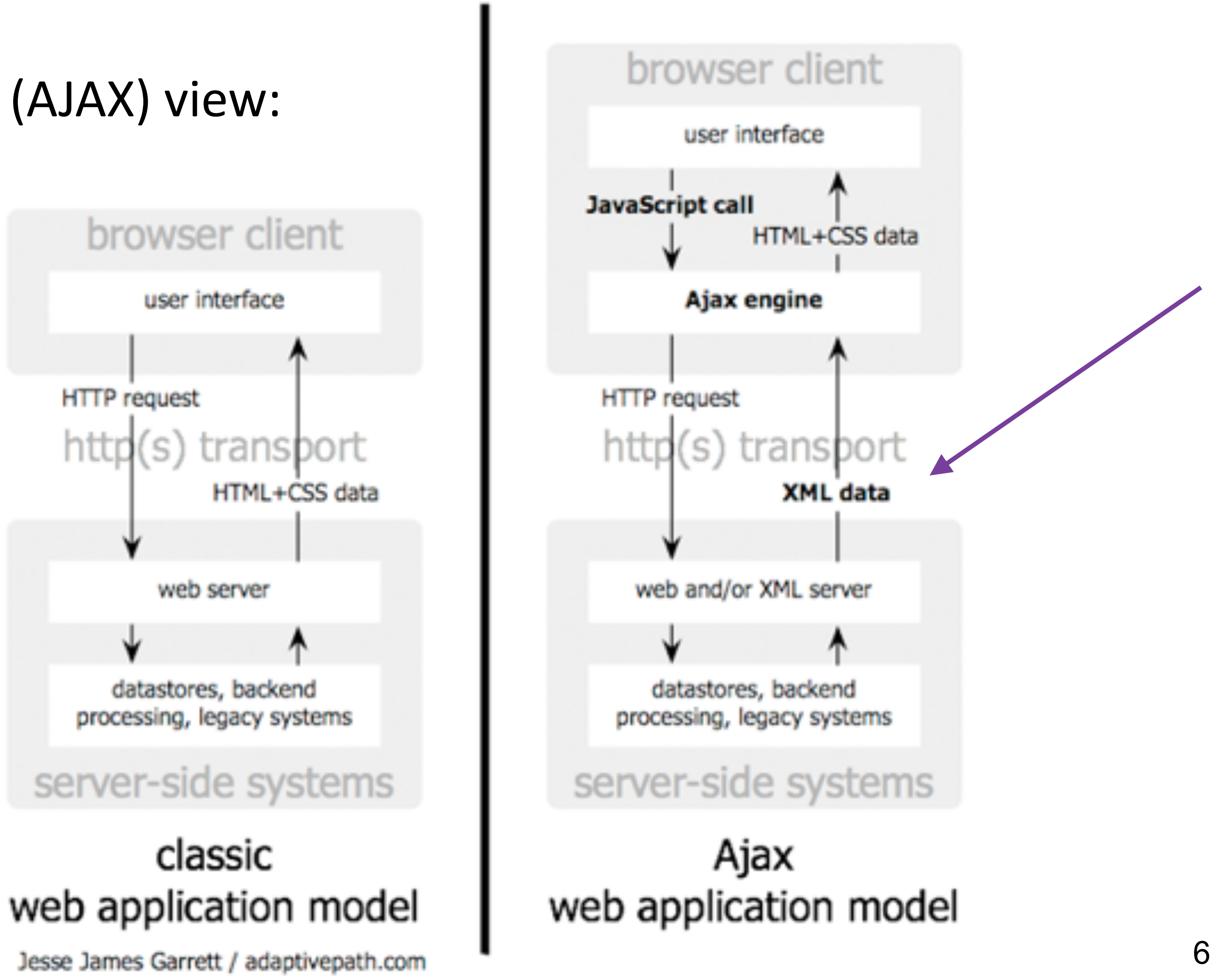
Example applications that generate or consume XML documents

- our fictional cartoon web site (Dilbert!)
  - submit new cartoon incl XML document describing it
  - search for cartoons
- an arithmetic learning web site (see CW2 in combination with CW1)
- a real learning site: Blackboard uses XML as a format to exchange information from your web browser to the BB server
  - student enrolment, coursework, marks & feedback, …
- RSS feeds:
  - hand-craft your own RSS channel or
  - build it automatically from other sources



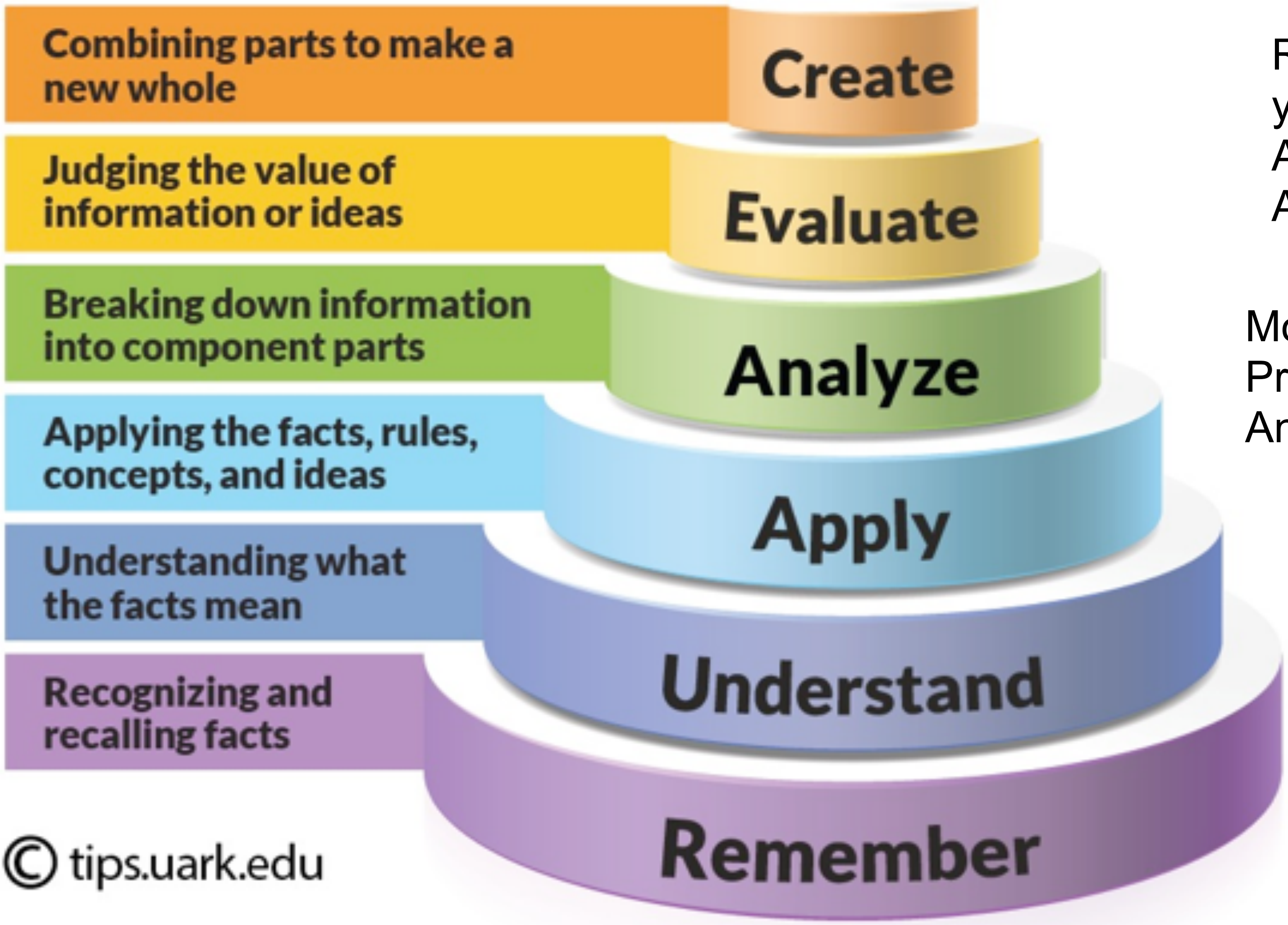
SE2 General Feedback: applications using XML

- Another (AJAX) view:



A Taxonomy of Learning

Your MSc/PhD Project


Combining parts to make a new whole
Create
Judging the value of information or ideas
Evaluate
Breaking down information into component parts
Analyze
Applying the facts, rules, concepts, and ideas
Apply
Understanding what the facts mean
Understand
Recognizing and recalling facts
Remember
© tips.uark.edu

Reflecting on your Experience, Answering SEx Analyze

Modelling, Programming, Answering Mx, CWx

Reading, Writing Glossaries Answering Qx

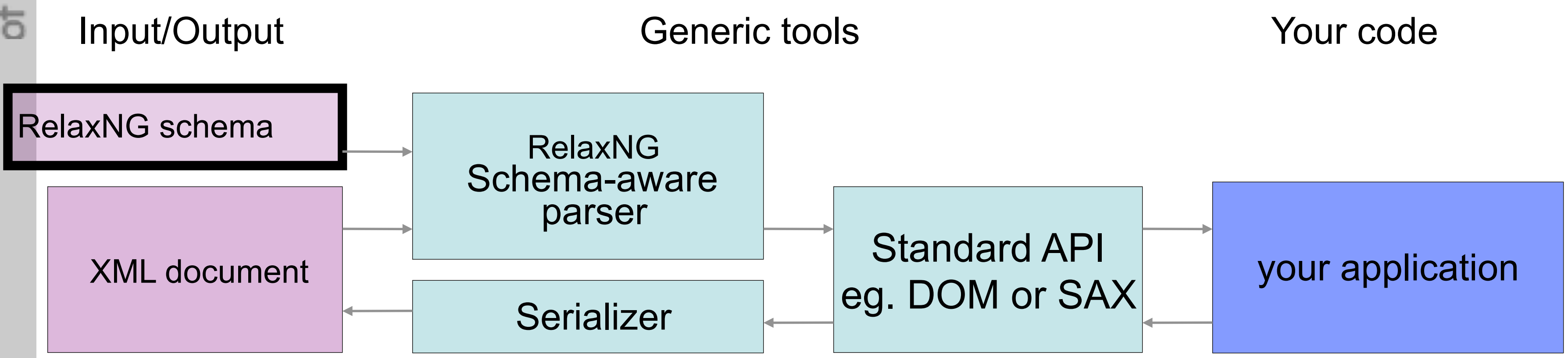
Today

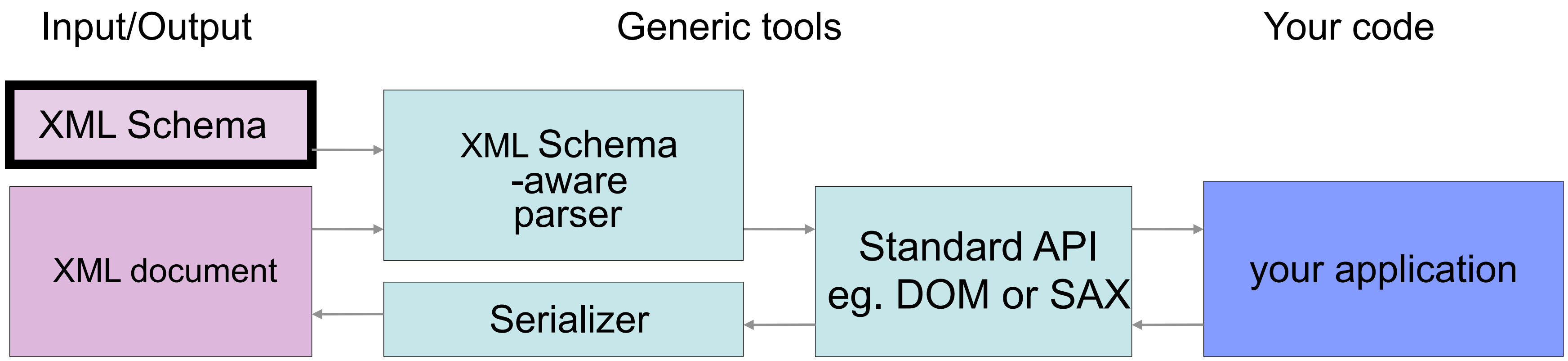
- SAX
  - alternative to DOM
  - an API to work with XML documents
  - parse & serialise
- Schematron
  - alternative to DTDs, RelaxNG, XSD
  - an XPath, error-handling oriented schema language
- JSON
  - alternative to XML

- More on
  - Errors & Robustness
  - Self-describing & Round-tripping

SAX

Remember: XML APIs/manipulation mechanisms





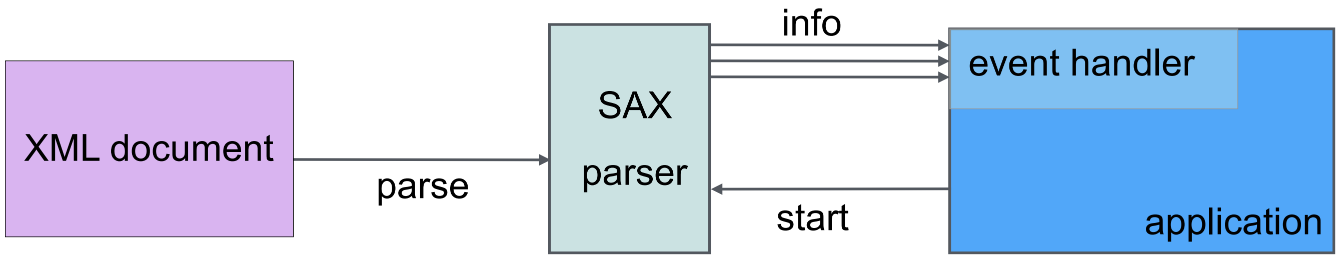
SAX parser in brief

- “SAX” is short for Simple API for XML
- not a W3C standard, but “quite standard”
- there is SAX and SAX2, using different names
- originally only for Java, now supported by various languages
- can be said to be based on a parser that is
  - multi-step, i.e., parses the document step-by-step
  - push, i.e., the parser has the control, not the application
    a.k.a. event-based

- in contrast to DOM,
  - **no parse tree is generated**/maintained
    ➥ useful for large documents
  - it has no generic object model
    ➥ no objects are generated & trashed
  - …remember SE2:
    - a good case mentioned often was:
      “we are only interested in a small chunk of the given XML document”
    - why would we want to build/handle whole DOM tree
      if we only need small sub-tree?

SAX in brief

- how the parser (or XML reader) is in control and the application "listens"



- SAX creates a series of events based on its depth-first traversal of document
- E.g.,

```
<?xml version="1.0" encoding="UTF-8"?>
<mytext content="medium">
        <title>
                Hallo!
        </title>
        <content>
                Bye!
        </content>
</mytext>
```

start document
start Element: mytext *attribute* content *value* medium
start Element: title
characters: Hallo!
end Element: title
start Element: content
characters: Bye!
end Element: content
end Element: mytext

SAX in brief

- SAX parser, when started on document D, goes through D while commenting what it does
- application listens to these comments, i.e., to list of all pieces of an XML document
  - whilst taking notes: when it's gone, it's gone!
- the primary interface is the ContentHandler interface
  - provides methods for relevant structural types in an XML document, e.g. startElement(), endElement(), characters()
- we need implementations of these methods:
  - we can use DefaultHandler
  - we can create a subclass of DefaultHandler and re-use as much of it as we see fit
- let's see a trivial example of such an application... from http://www.javaworld.com/javaworld/jw-08-2000/jw-0804-sax.html?page=4

```
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import java.io.*;
public class Example extends DefaultHandler {
   // Override methods of the DefaultHandler
  // class to gain notification of SAX Events.
public void startDocument( ) throws SAXException {
      System.out.println( "SAX E.: START DOCUMENT" );
   }

public void endDocument( ) throws SAXException {
      System.out.println( "SAX E.: END DOCUMENT" );
   }

public void startElement(
            String namespaceURI,
            String localName,
            String qName,
            Attributes attr ) throws SAXException {
     System.out.println( "SAX E.: START ELEMENT[ " +
          localName + " ]" );
   // and let's print the attributes!
        for ( int i = 0; i < attr.getLength(); i++ ){
            System.out.println( "   ATTRIBUTE: " +
            attr.getLocalName(i) + " VALUE: " +
            attr.getValue(i) );
      }
   }
```

NS!

```
 public void endElement(
            String namespaceURI,
            String localName,
            String qName ) throws SAXException {
   System.out.println( "SAX E.: END ELEMENT[ "localName + " ]" );
   }

public void characters( char[] ch, int start, int length )
               throws SAXException {
   System.out.print( "SAX Event: CHARACTERS[ " );
   try {
      OutputStreamWriter outw = new OutputStreamWriter(System.out);
      outw.write( ch, start,length );
      outw.flush();
   } catch (Exception e) {
      e.printStackTrace();
   }
   System.out.println( " ]" );
  }

public static void main( String[] argv ){
   System.out.println( "Example1 SAX E.s:" );
   try {
      // Create SAX 2 parser...
      XMLReader xr = XMLReaderFactory.createXMLReader();
      // Set the ContentHandler...
      xr.setContentHandler( new Example() );
        // Parse the file...
      xr.parse( new InputSource( new FileReader( ”myexample.xml" )));
   }catch ( Exception e )  {
      e.printStackTrace();
   }
  }
}
```

The ▨ parts are to be replaced with something more sensible, e.g.:

```
 if ( localName.equals( "FirstName" ) ) {
         cust.firstName = contents.toString();
...
```

SAX by example

- when applied to

```
<?xml version="1.0" encoding="UTF-8"?>
<uli:simple xmlns:uli="www.sattler.org" date="7/7/2000" >
   <uli:name DoB="6/6/1988" Loc="Manchester"> Bob </uli:name>
   <uli:location> New York </uli:location>
</uli:simple>
```

- this program results in

```
SAX E.: START DOCUMENT
SAX E.: START ELEMENT[ simple ]
  ATTRIBUTE: date VALUE: 7/7/2000
SAX Event: CHARACTERS[
    ]
SAX E.: START ELEMENT[ name ]
  ATTRIBUTE: DoB VALUE: 6/6/1988
  ATTRIBUTE: Loc VALUE: Manchester
SAX Event: CHARACTERS[  Bob  ]
SAX E.: END ELEMENT[ name ]
SAX Event: CHARACTERS[
    ]
SAX E.: START ELEMENT[ location ]
SAX Event: CHARACTERS[  New York  ]
SAX E.: END ELEMENT[ location ]
SAX Event: CHARACTERS[
 ]
SAX E.: END ELEMENT[ simple ]
SAX E.: END DOCUMENT
```

SAX: some pros and cons

+ fast: we don't need to wait until XML document is parsed before we can start doing things
+ memory efficient:
  the parser does not keep the parse/DOM tree in memory

+/-we might create our own structure anyway, so why duplicate effort?!

- we cannot "jump around" in the document; it might be tricky to keep track of the document's structure
- unusual concept, so it might take some time to get used to using a SAX parser

DOM and SAX -- summary

- so, if you are developing an application that needs to extract information from an XML document, you have the choice:
  - write your own XML reader
  - use some other XML reader
  - use DOM
  - use SAX
  - use XQuery
- all have pros and cons, e.g.,
  - might be time-consuming but may result in something really efficient because it is application specific
  - might be less time-consuming, but is it portable? supported? re-usable?
  - relatively easy, but possibly memory-hungry
  - a bit tricky to grasp, but memory-efficient

Back to Self-Describing

&

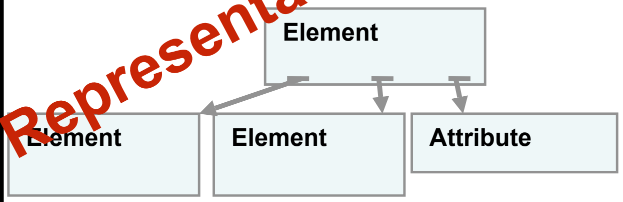
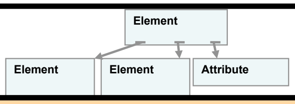
Different styles of schemas

The Essence of XML

cool paper mentioned in Week 2

- Thesis:
  - "XML is touted as an external format for representing data."
- Two properties
  - Self-describing
    - Destroyed by external validation,
    - i.e., using application-specific schema for validation, one that isn't referenced in the document
  - Round-tripping
    - Destroyed by defaults and union types

http://bit.ly/essenceOfXML2

Level		Data unit examples	Information or Property required
cognitive			
application			
tree adorned with...			
namespace	schema		nothing | a schema
tree			well-formedness
token	complex	<foo:Name t=”8”>Bob	
	simple	<foo:Name t=”8”>Bob	
character		<foo:Name t=”8”>Bob	which encoding (e.g., UTF-8)
bit		10011010	

Internal Representation

External Representation

erase

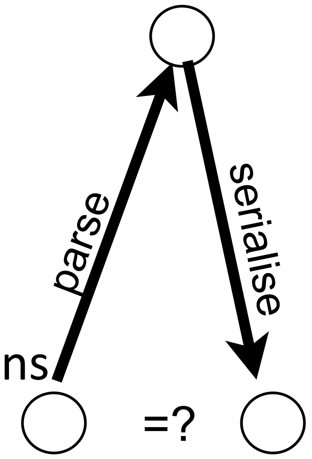
serialise

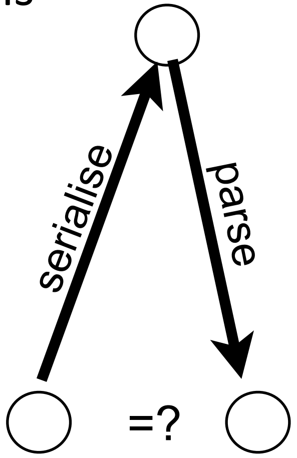
validate

parse

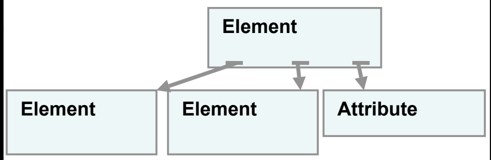
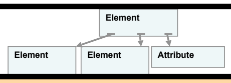
Roundtripping

- Within a single system:
  - roundtripping (both ways) should be *exact*
  - same program should behave the same in similar conditions
- Within various copies of the same systems:
  - roundtripping (both ways) should be *exact*
  - same program should behave the same in similar conditions
  - for interoperability!
- Within different systems
  - e.g., browser/client - server
  - roundtripping should be *reasonable*
  - analogous programs should behave analogously
  - in analogous conditions
  - a weaker notion of interoperability





What again is an XML document?

Level		Data unit examples	Information or Property required
cognitive			
application			
tree adorned with... namespace	schema		nothing / a schema
tree			well-formedness
token	complex	<foo:Name t=”8”>Bob	
	simple	<foo:Name t=”8”>Bob	
character		< foo:Name t=”8”>Bob	which encoding (e.g., UTF-8)
bit		10011010	

PSVI, Types, default values

Errors here ➜ no DOM!

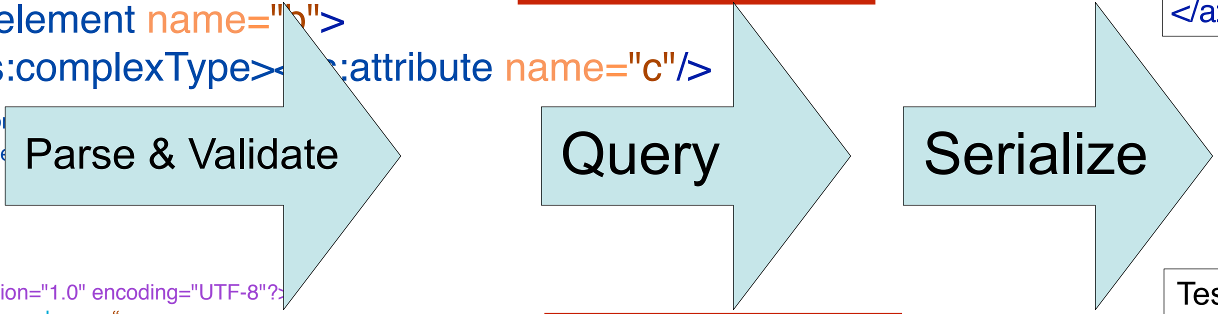
Roundtripping Fail: Defaults in XSD

sparse.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="…  >
<xs:element name="a">
   <xs:complexType><xs:sequence>
      <xs:element maxOccurs="unbounded" ref="b"/>
    </xs:sequence></xs:complexType>
 </xs:element>
 <xs:element name="b">
   <xs:complexType><xs:attribute name="c"/>
   </xs:co
</xs:eleme
```

Test.xml

```
<a>
   <b/>
   <b c="bar"/>
</a>
```



count(//@c) = ??

Test-sparse.xml

```
<a>
   <b/>
   <b c="bar"/>
</a>
```

count(//@c) = ??

Test-full.xml

```
<a>
   <b c="foo"/>
   <b c="bar"/>
</a>
```

full.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="…  >
<xs:element name="a">
   <xs:complexType><xs:sequence>
      <xs:element maxOccurs="unbounded" ref="b"/>
    </xs:sequence></xs:complexType>
 </xs:element>
 <xs:element name="b">
   <xs:complexType><xs:attribute name="c" default="foo"/>
   </xs:complexType>
</xs:element></xs:schema>
```

only diff!

Can we think of Test-sparse and -full as “the same”?

XML is not (always) self-describing!

- Under external validation
- Not just legality, but content!
  - The PSVIs have different information in them!

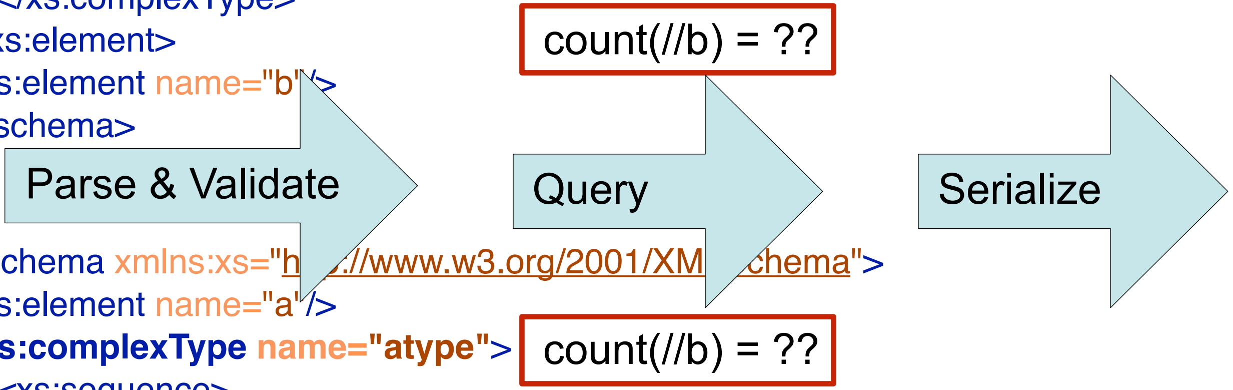
Roundtripping “Success”: Types

bare.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="a">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="b"  maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="b"/>
</xs:schema>
```

Test.xml

```
<a>
   <b/>
   <b/>
</a>
```



Test.xml

```
<a>
   <b/>
   <b/>
</a>
```

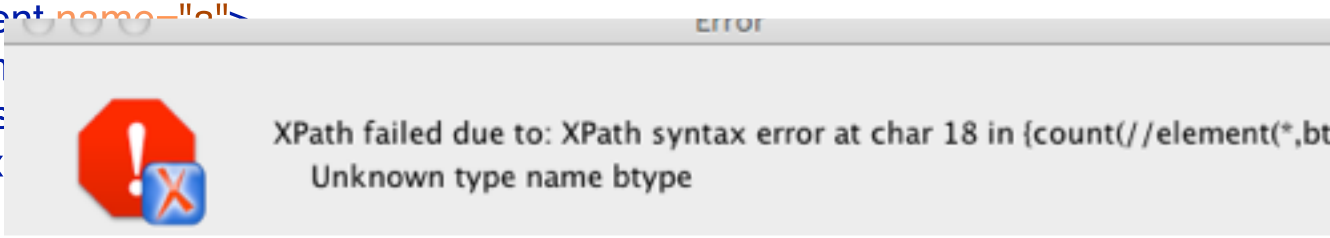
typed.xsd

```
<xs:schema xmlns:xs="h   ://www.w3.org/2001/XM   chema">
   <xs:element name="a"/>
   <xs:complexType name="atype">
      <xs:sequence>
         <xs:element ref="b"
                 maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
   <xs:element name="b" type="btype"/>
   <xs:complexType name="btype"/>
</xs:schema>
```

Roundtripping “Issue”: Types

bare.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="a">
    <xs:con
      <xs:s
        <x
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="b"/>
</xs:schema>
```

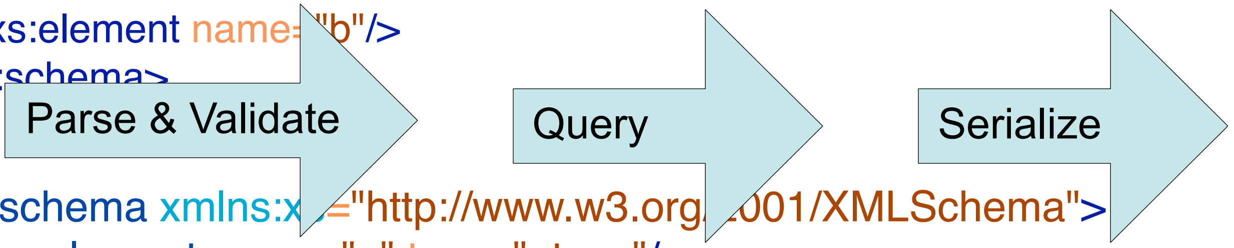

Error

XPath failed due to: XPath syntax error at char 18 in {count(//element(*,bt
Unknown type name btype

count(//element(*,btype)) = ??

Test.xml

```
<a>
  <b/>
  <b/>
</a>
```



Test.xml

```
<a>
  <b/>
  <b/>
</a>
```

typed.xsd

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="a" type="atype"/>
  <xs:complexType name="atype">
    <xs:sequence>
      <xs:element ref="b" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
  <xs:element name="b" type="btype"/>
  <xs:complexType name="btype"/>
</xs:schema>
```

count(//element(*,btype)) = ??

The Essence of XML

- Thesis:
  - “XML is touted as an external format for representing data.”
- Two properties
  - Self-describing
    - Destroyed by external validation,
    - i.e., using application-specific schema for validation
  - Round-tripping
    - Destroyed by defaults and union types

http://bit.ly/essenceOfXML2

An Excursion into JSON - another tree data structure formalism: the *fat-free* alternative to XML

http://www.json.org/xml.html

JavaScript Object Notation

- JSON was developed to serialise/store/transmit/… JavaScript objects
  - other programming languages can read/write JSON as well
  - (just like XML)
- Given some J objects we can serialise them into
  - XML: involves design choices
    - attribute or child element?
    - element/attribute names?
  - JSON: basically automatic

MANCHESTER 1824

The University of Manchester

JavaScript Object Notation - JSON

- Javascript has a rich set of literals (ext. reps) called items
  - **Atomic** (numbers, booleans, strings*)
    - 1, 2, true, “I’m a string”
  - **Composite**
    - **Arrays**
      - Ordered lists with random access
      - e.g., [1, 2, “one”, “two”]
    - “**Objects**”
      - Sets/unordered lists/associative arrays/dictionary
      - {“one”:1, “two”:2}
  - these can nest!
    - [{“one”:1, “o1”:{“a1”: [1,2,3.0], “a2”:[]}]
- JSON = roughly this subset of Javascript
- The internal representation varies
  - In JS, 1 represents a 64 bit, IEEE floating point number

Note: {….} is a set
[…] is a list/array

JSON - XML example

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  }
}}
```

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

slightly different

JSON - XML example

order matters!

```
{"menu": {
  "id": "file",
  "value": "File",
  "popup": [
    "menuitem": [
      {"value": "New", "onclick": "CreateNewDoc()"},
      {"value": "Open", "onclick": "OpenDoc()"},
      {"value": "Close", "onclick": "CloseDoc()"}
    ]
  ]
}}
```

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

less different!

MANCHESTER
1824

The University of Manchester

JSON - XML example

attribute nodes!

```
{"menu": [{"id": "file", "value": "File"},
   [{"popup":
            [{},
            [{"menuitem": [{"value": "New", "onclick": "CreateNewDoc()"},[]]},
             {"menuitem": [{"value": "Open", "onclick": "OpenDoc()"},[]]},
             {"menuitem": [{"value": "Close", "onclick": "CloseDoc()"},[]]}
            ]
            ]
           }
          ]
         ]
}
```

```
<menu id="file" value="File">
   <popup>
      <menuitem value="New" onclick="CreateNewDoc()" />
      <menuitem value="Open" onclick="OpenDoc()" />
      <menuitem value="Close" onclick="CloseDoc()" />
   </popup>
</menu>
```

even more similar!

XML ➟ JSON (a recipe)

- each element is mapped to an “object”
  - consisting of a single pair {ElementName : contents}
- contents is a list
  - 1st item is an “object” ({…}, unordered) for the attributes
    - attributes are pairs of strings
    - e.g., {"id": "file", "value": "File"}
  - 2nd item is an array ([…], ordered) for child elements

```
<a>
    <b id="1" type="Fun"/>
    <b id="2"/>
</a>
```

```
{a:[{},
    {b:[{“id”:“1”, “type”:“Fun”},[] ]}
    {b:[{“id”:“2”,[] ]}
]}
```

- Empty elements require an explicit empty list
- No attributes requires an explicit empty object

True or False?

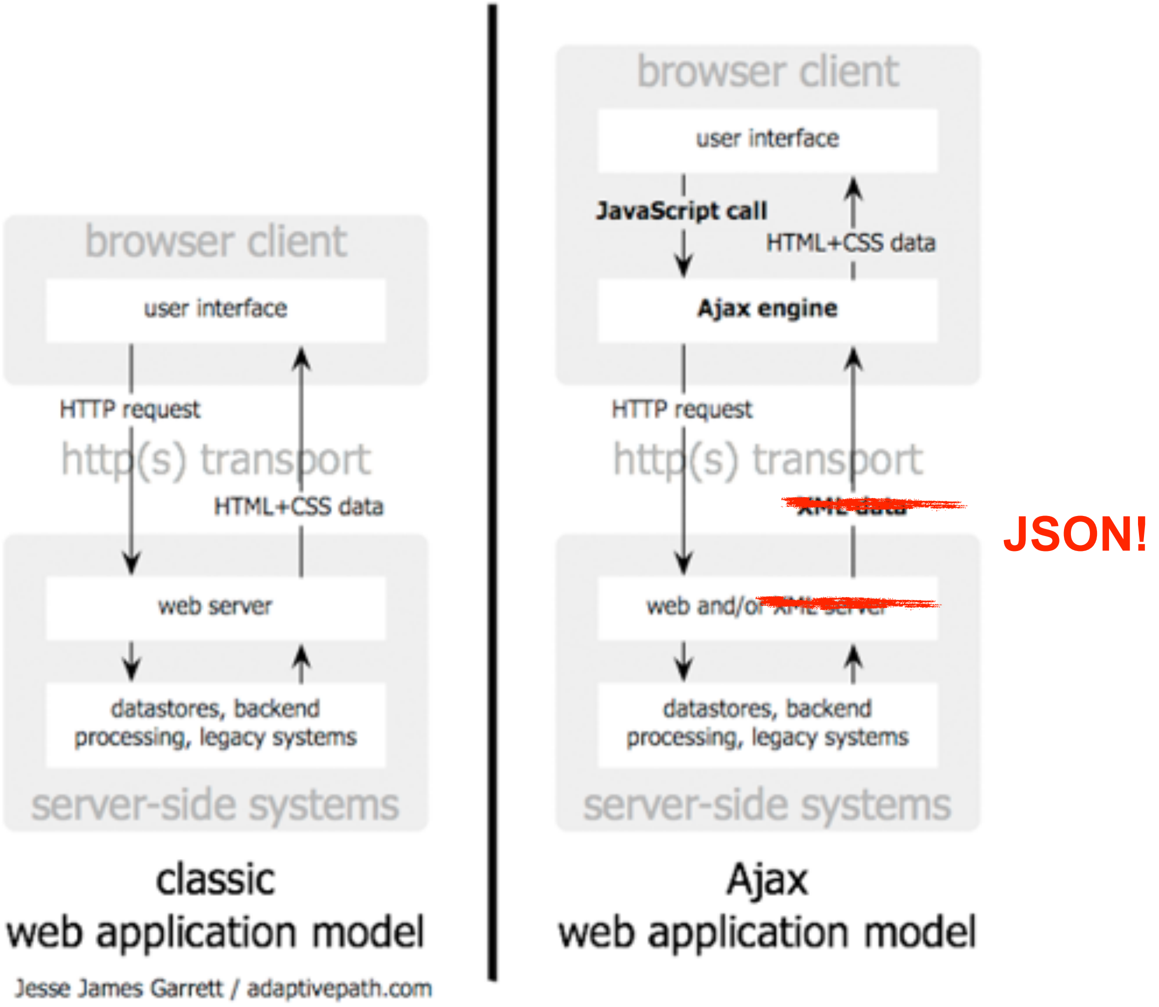
1. Every **JSON item** can be *faithfully* represented as an **XML document**
2. Every **XML document** can be *faithfully* represented as a **JSON item**
3. Every **XML DOM** can be *faithfully* represented as a **JSON item**
4. Every J**SON item** can be *faithfully* represented as an **XML DOM**
5. Every **WXS PSVI** can be *faithfully* represented as a **JSON item**
6. Every **JSON item** can be *faithfully* represented as a **WXS PSVI**

Affordances

- Mixed Content
  - XML
    - `<p><em>Hi</em> there!</p>`
  - JSON

    -
      ```
      {"p": [
          {"em": "Hi"},
          "there!"
      ]}
      ```
  - Not great for hand authoring!
- Config files
- Anything with integers?
- Simple processing
  - XML:
    - DOM of Doom, SAX of Sorrow
    - Escape to query language
  - JSON
    - Dictionaries and Lists!

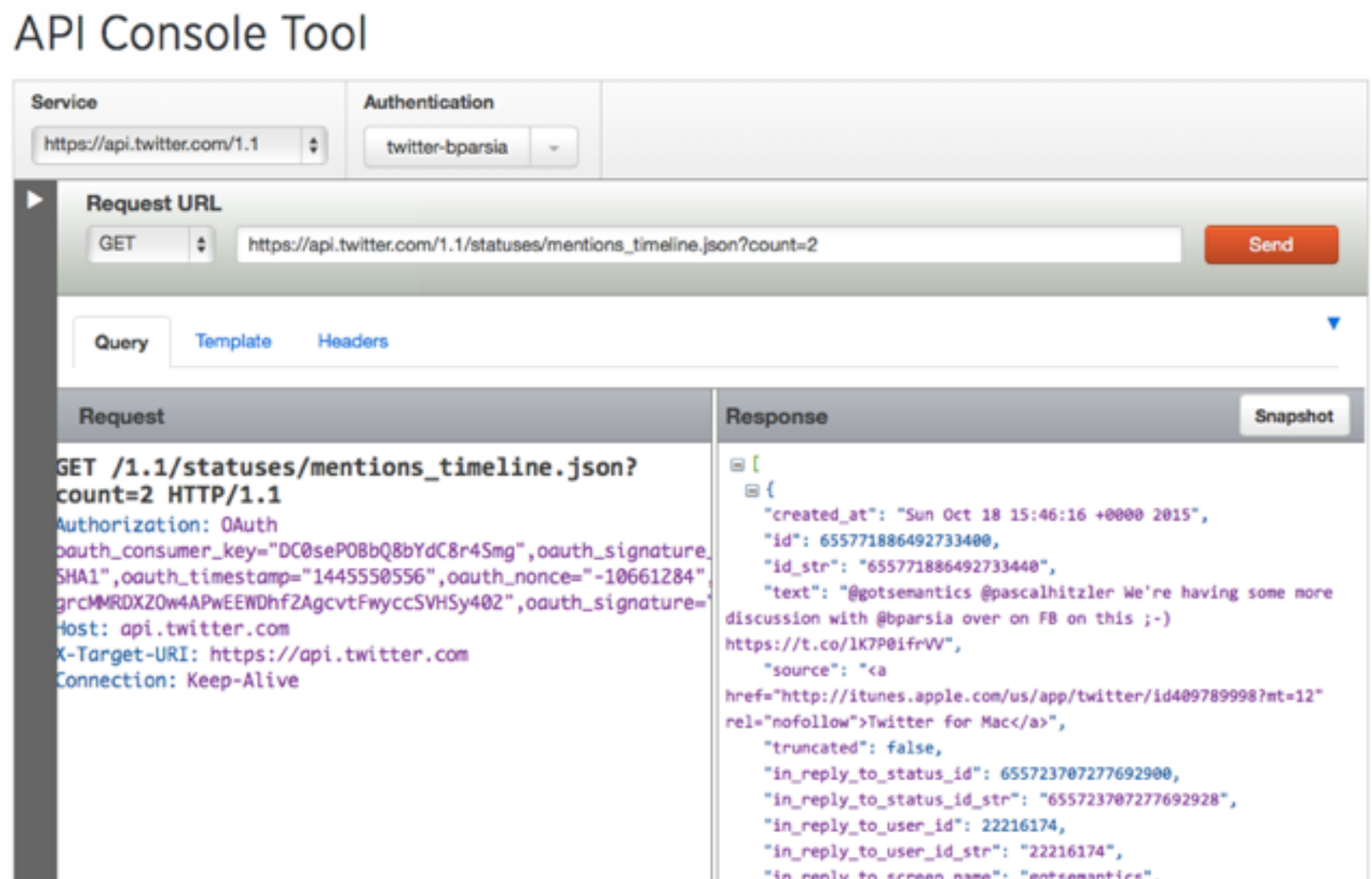
Applications using ~~XML~~ JSON!



Try it: http://jsonplaceholder.typicode.com

Twitter Demo

- https://dev.twitter.com/rest/tools/console



API Console Tool

Service: https://api.twitter.com/1.1

Authentication: twitter-bparsia

Request URL

GET https://api.twitter.com/1.1/statuses/mentions_timeline.json?count=2 Send

Query Template Headers

Request

```
GET /1.1/statuses/mentions_timeline.json?
count=2 HTTP/1.1
Authorization: OAuth
oauth_consumer_key="DC0sePOBbQ8bYdC8r4Smg",oauth_signature_
SHA1",oauth_timestamp="1445550556",oauth_nonce="-10661284",
grcMMRDXZOw4APwEEWDhf2AgcvtFwyccSVHSy402",oauth_signature="
Host: api.twitter.com
X-Target-URI: https://api.twitter.com
Connection: Keep-Alive
```

Response Snapshot

```
[
  {
    "created_at": "Sun Oct 18 15:46:16 +0000 2015",
    "id": 655771886492733400,
    "id_str": "655771886492733440",
    "text": "@gotsemantics @pascalhitzler We're having some more
discussion with @bparsia over on FB on this ;-)
https://t.co/1K7P0ifrVV",
    "source": "<a
href="http://itunes.apple.com/us/app/twitter/id409789998?mt=12"
rel="nofollow">Twitter for Mac</a>",
    "truncated": false,
    "in_reply_to_status_id": 655723707277692900,
    "in_reply_to_status_id_str": "655723707277692928",
    "in_reply_to_user_id": 22216174,
    "in_reply_to_user_id_str": "22216174",
    "in_reply_to_screen_name": "gotsemantics",
```

Is JSON edging towards SQL completeness?

- Do we have (even post-facto) schemas?
  - Historically, mostly code
  - But there have been schema proposals, such as
    - json-schema
      - http://spacetelescope.github.io/understanding-json-schema/
      - http://jsonschema.net/#/
- Json-schema
  - Rather simple!
  - Simple patterns
    - Types on values (but few types!)
    - Some participation/cardinality constraints (allOf, oneOf,..)
    - Lexical patterns
      - Email addresses!

Example

- http://json-schema.org/example1.html

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "Product",
    "description": "A product from Acme's catalog",
    "type": "object",
    "properties": {
        "id": {
            "description": "The unique identifier for a product",
            "type": "integer"
        },
        "name": {
            "description": "Name of the product",
            "type": "string"
        },
        "price": {
            "type": "number",
            "minimum": 0,
            "exclusiveMinimum": true
        }
    },
    "required": ["id", "name", "price"]
}
```

JSON *Databases*?

- NoSQL “movement”
  - Originally “throw out features”
    - Still quite a bit
  - Now, a bit of umbrella term for semi-structured databases
    - So XML counts!
  - Some subtypes:
    - Key-Value stores
    - Document-oriented databases
    - Graph databases
    - Column databases
- Some support JSON as a layer
  - E.g., BaseX
- Some are “JSON native”
  - MongoDB
  - CouchDB

MANCHESTER
1824

The University of Manchester

Error Handling

Errors - everywhere & unavoidable!

- E.g., CW3 - what to do for (7 + 9)/(3 - (1 + 2))?
- Preventing errors: make
  - errors hard or impossible to make
    - Make doing things hard or impossible
  - doing the right thing easy and inevitable
  - detecting errors easy
  - correcting errors easy

- Correcting errors:
  - fail silently
  - ? Fail randomly
  - ? Fail differently (interop problem)

Postel's Law

Be liberal in what you accept,
and
conservative in what you send.

- Liberality
  - Many DOMs, all expressing the same thing
  - Many surface syntaxes (perhaps) for each DOM
- Conservativity
  - What *should* we send?
    - It depends on the receiver!
  - Minimal standards?
    - Well formed XML?
    - Valid according to a popular schema/format?
    - HTML?

Error Handling - Examples

- XML has **draconian** error handling
  - 1 Well-formedness error…BOOM
- CSS has **forgiving** error handling
  - "Rules for handling parsing errors"
    http://www.w3.org/TR/CSS21/syndata.html#parsing-errors
    - That is, how to *interpret* illegal documents
    - Not reporting errors, but working around them
  - e.g.,"User agents must ignore a declaration with an unknown property."
    - Replace: "`h1 { color: red; rotation: 70minutes }`"
    - With: "`h1 { color: red }`"
- Check out CSS's error handling rules!

XML Error Handling

- De facto XML motto
  - be strict about the well-formed-ness of what you accept,
  - and strict in what you send
  - Draconian error handling
  - Severe consequences on the Web
    - And other places
- Fail early and fail hard
- What about higher levels?
  - Validity and other analysis?
  - Most schema languages are poor at error reporting
    - How about XQuery’s type error reporting?
    - XSD schema-aware parser report on
      - error location (which element) and
      - what was expected
      - …so we could fix things!?

Typical Schema Languages

- Grammar (and maybe type based)
  - Validation: either succeeds or FAILs
  - Restrictive by default: what is not permitted is forbidden
    - what happens in this case?

```
element a { attribute value { text }, empty }
```

```
<a value="3" date="2014"/>
```

  - Error detection and reporting
    - Is at the *discretion* of the system
    - “Not accepted” **may** be the only answer the validator gives!
    - The point where an error is detected
      - might not be the point where it occurred
      - might not be the most helpful point to look at!
    - Compare to programs!
      - Null pointer deref
        - » Is the right point the deref or the setting to null?

MANCHESTER 1824

The University of Manchester

Our favourite Way

Be liberal in what you accept,
and
conservative in what you send.

- Adore Postel's Law
- Explore before prescribe
- Describe rather than define
- Take what you can, when/if you can take it
  - don't be a horrible person/program/app!
- Design your **formats** so that extra or missing stuff is (can be) OK
  - Irregular structure!
- Adhere to the task at hand

How many middle/last/first names does your address **format** have?!

MANCHESTER 1824

The University of Manchester

XPath for Validation

- Can we use XPath to determine constraint violations?

simple.rnc

```
grammar {
start     = element a { b-descr+ }
b-descr = element b { empty} }
```

valid.xml

```
<a>
   <b/>
   <b/>
   <b/>
</a>
```

count(//b)	count(//b/*)	count(//b/text())
=3 ✔	=0 ✔	=0 ✔
✔ =4	✗ =1	✗ =1

invalid.xml

```
<a>
   <b/>
   <b>Foo</b>
   <b><b/></b>
</a>
```

```
<a>
   <b/>
   <b>Foo</b>
</a>
```

✔ =0

```
<a>
   <b/>
   <b><b/><b/>
</a>
```

✔ =0

XPath for Validation

- Can we use XPath to determine constraint violations?

simple.rnc

```
grammar {
start    = element a { b-descr+ }
b-descr = element b { empty} }
```

valid.xml

```
<a>
   <b/>
   <b/>
   <b/>
</a>
```

=0 ✔

count(//b/(* | text()))

invalid.xml

```
<a>
   <b/>
   <b>Foo</b>
   <b><b/></b>
</a>
```

✗ =2

```
<a>
   <b/>
   <b>Foo</b>
</a>
```

✗ =1

```
<a>
   <b/>
   <b><b/><b/>
</a>
```

✗ =1

Yes!

XPath for Validation

- Can we use XPath to determine constraint violations?

simple.rnc

```
grammar {
start     = element a { b-descr+ }
b-descr = element b { empty} }
```

valid.xml

```
<a>
   <b/>
   <b/>
   <b/>
</a>
```

= valid

invalid.xml

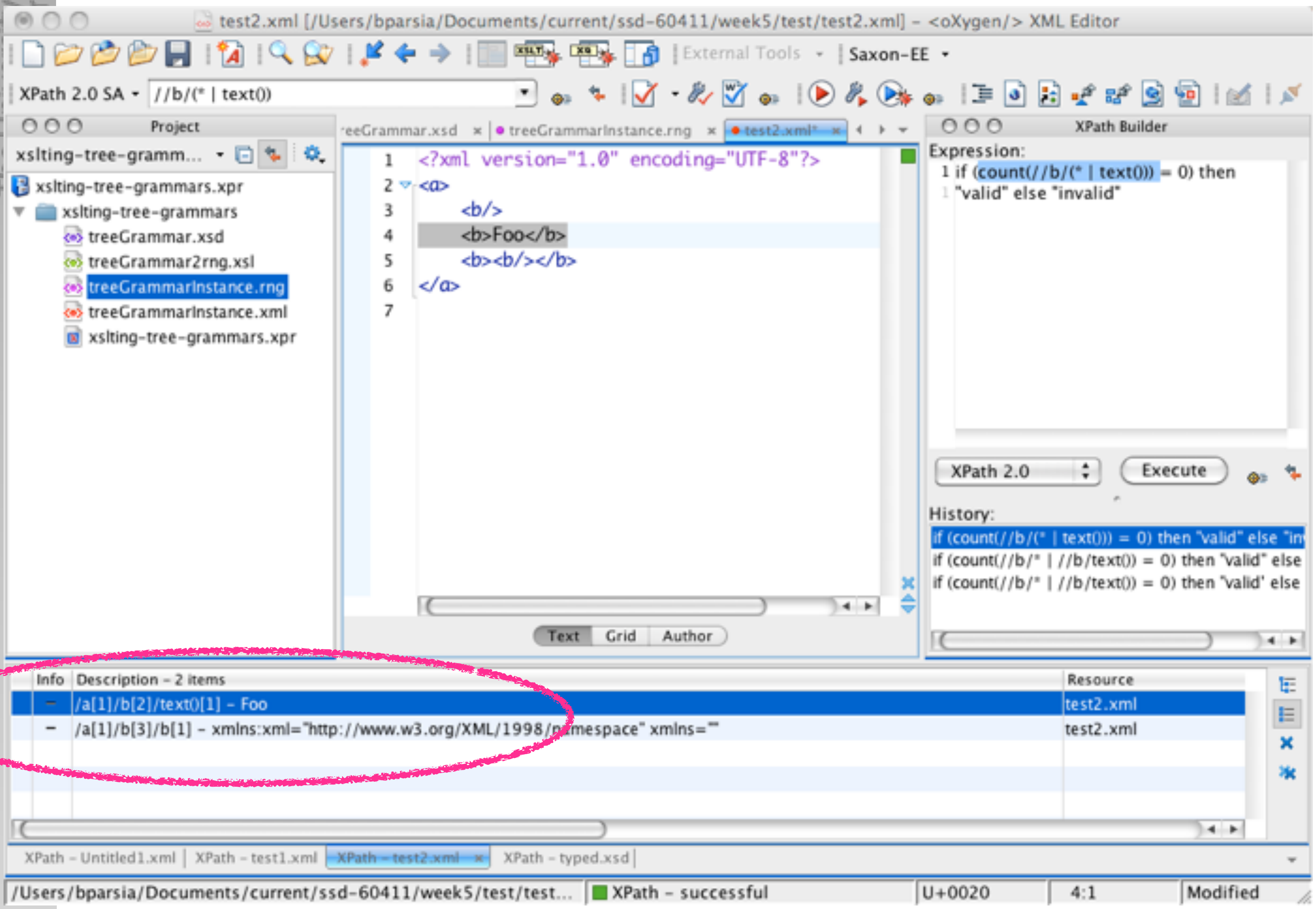
```
<a>
   <b/>
   <b>Foo</b>
   <b><b/></b>
</a>
```

= invalid

```
<a>                        <a>
   <b/>                       <b/>
   <b>Foo</b>                 <b><b/><b/>
</a>                       </a>
```

```
if (count(//b/(* | text()))=0)
   then “valid”
   else “invalid”
```

Can even “locate” the errors!



test2.xml [/Users/bparsia/Documents/current/ssd-60411/week5/test/test2.xml] - <oXygen/> XML Editor

XPath 2.0 SA `//b/(* | text())`

Project: xslting-tree-gramm...

- xslting-tree-grammars.xpr
- xslting-tree-grammars
  - treeGrammar.xsd
  - treeGrammar2rng.xsl
  - treeGrammarInstance.rng
  - treeGrammarInstance.xml
  - xslting-tree-grammars.xpr

```xml
<?xml version="1.0" encoding="UTF-8"?>
<a>
    <b/>
    <b>Foo</b>
    <b><b/></b>
</a>
```

XPath Builder

Expression:

```
if (count(//b/(* | text())) = 0) then
"valid" else "invalid"
```

XPath 2.0 | Execute

History:

```
if (count(//b/(* | text())) = 0) then "valid" else "in
if (count(//b/* | //b/text()) = 0) then "valid" else
if (count(//b/* | //b/text()) = 0) then 'valid' else
```

Info	Description - 2 items	Resource
–	/a[1]/b[2]/text()[1] - Foo	test2.xml
–	/a[1]/b[3]/b[1] - xmlns:xml="http://www.w3.org/XML/1998/namespace" xmlns=""	test2.xml

XPath - Untitled1.xml | XPath - test1.xml | XPath - test2.xml | XPath - typed.xsd

/Users/bparsia/Documents/current/ssd-60411/week5/test/test... | XPath - successful | U+0020 | 4:1 | Modified

XPath (etc) for Validation

- We could have finer control
  - Validate parts of a document
  - A la wildcards
    - But with more control!
- We could have high expressivity
  - Far reaching dependancies
  - Computations
- Essentially, code based validation!
  - With XQuery and XSLT
  - But still a little declarative
- We always need it

The essence of Schematron

Schematron

Schematron

- A different sort of schema language
  - **Rule** based
    - **Not** grammar based or object/type based
  - **Test** oriented
  - **Complimentary** to other schema languages
- Conceptually **simple**: patterns contain rules
  - a rule sets a **context** and contains
    - **asserts** (As) - act "when test is false"
    - **reports** (Rs) - act "when test is true"
  - A&Rs contain
    - a **test** attribute: XPath expressions, and
    - **text content**: natural language description of the error/issue

```
<assert test="count(//b/(*ltext())) = 0">
      Error: b elements must be empty
  </assert>
```

```
<report test="count(//b/(*ltext()))!= 0">
      Error b elements must be empty
  </report>
```

Schematron by example: for PLists

- "PList has at least 2 person child elements"

```
<pattern>
        <rule context="PList">
            <assert test="count(person) >= 2">
                There has to be at least 2 persons!
            </assert>
        </rule>
    </pattern>
```

- equivalently as a "report":

```
 <pattern>
        <rule context="PList">
            <report test="count(person) &lt; 2">
                There has to be at least 2 persons!
            </report>
        </rule>
    </pattern>
```

```
<PList>
    <person FirstName="Bob"
              LastName="Builder"/>
    <person FirstName="Bill"
              LastName="Bolder"/>
    <person FirstName="Bob"
              LastName="Builder"/>
</PList>
```

is valid w.r.t. these

```
<PList>
    <person FirstName="Bob"
              LastName="Builder"/>
  </PList>
```

is not valid w.r.t. these

Ok, could handle this with RelaxNG, XSD, DTDs…

Schematron by example: for PLists

- “Only 1 person with a given name”

```
<PList>
   <person FirstName="Bob"
              LastName="Builder"/>
   <person FirstName="Bill"
              LastName="Bolder"/>
   <person FirstName="Bob"
              LastName="Builder"/>
</PList>
```

```
<pattern>
      <rule context="person">
         <let name="F" value="@FirstName"/>
         <let name="L" value="@LastName"/>
         <assert test="count(//person[@FirstName = $F and @LastName = $L]) = 1">
            There can be only one person with a given name,
            but there is <value-of select="$F"/> <value-of select="$L"/> at least twice!
         </assert>
      </rule>
   </pattern>
```

above example is not valid w.r.t. these and causes nice error:

```
…
Engine name: ISO Schematron
Severity: error
Description: There can be only one person with a given name,
but there is Bob Builder at least twice!
```

Ok, could handle this with **Keys** in XML Schema!

MANCHESTER 1824

The University of Manchester

Schematron by example: for PLists

- “At least 1 person for each family”

```
<PList>
   <person FirstName="Bob" LastName="Builder"/>
   <person FirstName="Bill" LastName="Bolder"/>
   <person FirstName="Bob" LastName="Milder"/>
   <family name="Builder" town="Manchester"/>
   <family name="Bolder" town="Bolton"/>
</PList>
```

```
<pattern>
      <rule context="person">
         <let name="L" value="@LastName"/>
         <report test="count(//family[@name = $L]) = 0"> There has to be a
            family for each person mentioned, but
            <value-of select="$L"/> has none!
         </report>
      </rule>
   </pattern>
```

above example is not valid w.r.t. these and causes nice error:

```
…
Engine name: ISO Schematron
Severity: error
Description: There has to be a family for each person mentioned, but
Milder has none!
```

Schematron: informative error messages

```
<pattern>
     <rule context="person">
        <let name="L" value="@LastName"/>
        <report test="count(//family[@name = $L]) = 0">
          Each person's LastName must be declared in a family element!
       </report>
     </rule>
  </pattern>
```

If the `test` condition **true**, the content of the `report` element is displayed to the user.

```
<pattern>
     <rule context="person">
        <let name="L" value="@LastName"/>
        <report test="count(//family[@name = $L]) = 0"> There has to be a
           family for each person mentioned, but
           <value-of select="$L"/> has none!
        </report>
     </rule>
  </pattern>
```

Tip of the iceberg

- Computations
  - Using XPath functions and variables
- Dynamic checks
  - Can pull stuff from other file
- Elaborate reports
  - diagnostics has (value-of) expressions
  - “Generate paths” to errors
    - Sound familiar?
- General case
  - Thin shim over XSLT
  - Closer to “arbitrary code”

Schematron - Interesting Points

- Friendly: combine Schematron with WXS, RelaxNG, etc.
  - Schematron is good for that
  - Two phase validation
    - RELAX NG has a way of embedding
    - WXS 1.1 incorporating similar rules
- Powerful: arbitrary XPath for context and test
  - Plus variables
  - see M4!

Schematron - Interesting Points

- Lenient: what isn't forbidden is permitted
  - Unlike all the other schema languages!
  - We're not performing runs
    - We're firing rules
  - Somewhat easy to use
    - If you know XPath
    - If you don't need coverage
- No traces in **PSVI**: a document D either
  - passes all rules in a schema S
    - success -> D is valid w.r.t. S
  - fails some of the rules in S
    - failure -> D is not valid w.r.t. S
- …up to application what to do with D
  - possibly depending on the error messages…think of SE2

Schematron presumes…

- …well formed XML
  - As do all XML schema languages
    - Work on DOM!
  - So can't help with e.g., overlapping tags
    - Or tag soup in general
    - Namespace Analysis!?
- …authorial (i.e., human) repair
  - At least, in the default case
    - Communicate errors to people
    - Thus, not the basis of a modern browser!
      - Unlike CSS
- Is this enough liberality?
  - Or rather, does it support enough liberality?

MANCHESTER 1824

The University of Manchester

This Week's coursework

As usual…

- Quiz
- M4: write a Schematron schema that captures a given set of constraints
  - use an XML editor that supports Schematron (oxygen does)
  - make & share test cases on the forum!
  - work on simple cases first
  - read the tips!
- CW4: another XQuery one!
  - analyse namespaces
  - namespaces look like attributes but are different

As usual…

- SE4:
  - we ask you to discuss a **format**: does it use XML’s features well?
  - answer the question
  - think about properties we have mentioned in class!
  - is this format such that it is easy to
    - write conforming documents
    - avoid errrors
    - query it (using XQuery,…)
    - extend it to other pieces of information?
  - don’t repeat known points
  - structure your essay well
  - use a spell checker