Languages for Data Integration of Semi-Structured Data II – XML Schema, Dom/SAX

Recuperación de Información 2007

Lecture 3.

Overview

- XML-schema, a powerful alternative to DTDs
- XML APIs:
  - DOM, a data-object Model for XML
  - SAX, a simple API for XML
  - Validating Parsers based on DOM and SAX
- More XML Companion standards

XML Schema

- Another way to describe Structure of an XML document.

DTDs have very limited typing, missing:

- **Cardinalities** in DTDs hardly expressable… at least very inconvenient.
- Reuse, inheritance of **types** is missing, etc.
- Typing, Datatypes…

  … All this is provided in XML Schema!

XML-Schema : Why use XML-schema?

In order to validate an XML document you use XML Schema to specify:

- The allowed **structure** of an XML document
- The allowed **data types** contained in one (or several) XML documents

XML-Schema : Why use schemas instead of DTDs?

- XML Schema Documents (XSDs) themselves are XML documents and therefore use the same syntax and can themselves be validated with schemas! (whereas DTDs are somehow SGML "legacy")

- XML-schema has more powerful possibilites to define custom **datatypes** (regular expressions, inheritance, cardinalities etc.)

- (BTW: Schemas may also be combined with DTDs)

XML-Schema : Why use schemas instead of DTDs?

- XML-schema allows to define elements with nil content
- XML-schema allows to **reuse** element and attribute definitions (by reference)
- XML-schema **uses XML namespaces** (allows to refer to and prescribe certain namespaces)
- *XML-schema allows to define full context-free grammars for expressing arbitrary XML structures!*

XML-Schema : A Simple XML-Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<marketing
    xmlns:xsi = "http://www.w3c.org/2001/XMLSchema-Instance"
    xsi:noNameSpaceSchemaLocation = "http://www.Dot.com/mySchema.xsd">

  <employee>Gustav Sielmann</employee>
  <employee>Arnold Rummer</employee>
  <employee>Johann Neumeier</employee>

</marketing>
```

Instance

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd = "http://www.w3c.org/2001/XMLSchema">

<xsd:element name = "marketing>
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name = "employee" type = "xsd:string"
                   maxOccurs = "unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

</xsd:schema>
```

Schema Definition

General features of XSD (XML Schema Definition)

- XML Schema is always a separate, i.e. **external** entity (file)
  - No internal Schema within the xml-instance
- Schemalink via **attribute in Document Element**
  - **schemaLocation** attribute (for Schema with a tragetNamespace)
  - **noNameSpaceSchemaLocation** (for schema with no targetNamespace)

- Schemata can be embedded:
  - <include>, <redefine>, <import>
  - import if different namespaces
    <xsd:import schemaLocation=" http://www.w3.org/1999/xlink xlink.xsd"/>

    with <redefine> single elements can be redefined e.g. a second schema which redefines the first one:
    <xsd:redefine schemaLocation="..."><xsd:complexType ...../>
    which restricts or extends the type with the same name from the original schema

XSD and namespaces:

- XML Schema
  - *uses namespaces itself* - to distinguish schema instructions from the language we are describing
  - *supports namespace assigning* - by associating a **target namespace** to the **language** we are describing.

- Example:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:b="http://businesscard.org"
        targetNamespace="http://businesscard.org">

  <element name="card" type="b:card_type"/>
  ...

  <complexType name="card_type">
    <sequence>
      <element ref="b:name"/>
      ...
    </sequence>
  </complexType>

  ...
</schema>
```

Here:

- the **default** namespace is that of XML Schema (such that e.g. complexType is considered an XML Schema element)
- the **target** namespace is our business card namespace
- the b prefix also denotes our business card namespace (such that we can refer to target language constructs from within the schema)

XSD structure

An XSD is composed of:

- The Schema Element
- Element Definitions
- Attribute Definitions
- Type Definitions
- Annotations

XSD : The Schema Element

- The schema element is the container element where all elements, attributes and data types contained in an XML document are stored
- The schema element refers to the XML-schema definition at W3C and is the **document element (root)** of an XSD:

```
<xsd:schema xmlns:xsd = "http://www.w3c.org/2001/XMLSchema">
          .
          .
          .
          .
          .
</xsd:schema>
```

XSD: Schema element

Can have several attributes:

```
<?xml version="1.0"?>
<xsd:schema
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3schools.com"
  xmlns="http://www.w3schools.com"
  elementFormDefault="qualified">
...
</xsd:schema>
```

Elements and data types used in an XML schema document (schema, element, complexType, sequence, string, boolean, etc.) come from the "http://www.w3.org/2001/XMLSchema" namespace, and must be prefixed with xsd:

XSD: Schema element

Can have several attributes:

```
<?xml version="1.0"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3schools.com"
  xmlns="http://www.w3schools.com"
  elementFormDefault="qualified">
...
</xsd:schema>
```

The language defined by this schema (note, to, from, heading, body.) is supposed to be in the "http://www.w3schools.com" namespace.

XSD: Schema element

Can have several attributes:

```
<?xml version="1.0"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3schools.com"
  xmlns="http://www.w3schools.com"
  elementFormDefault="qualified">
...
</xsd:schema>
```

default namespace is "http://www.w3schools.com". Usually makes sense to have the same def.ns and targetns…

XSD: Schema element

Can have several attributes:

```
<?xml version="1.0"?>
<xsd:schema
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.w3schools.com"
  xmlns="http://www.w3schools.com"
  elementFormDefault="qualified">
...
</xsd:schema>
```

Any elements used by an **XML instance document** which were declared in this schema must be namespace qualified by the target namespace (analogous: attributeFormDefault), i.e. the language defined here is bound to the namespace. alternative: "unqualified",

How to reference XSD inside an instance document:

In the instance file:

```
<?xml version="1.0"?> <note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.w3schools.com
note.xsd">
```

Again default namespace, (this is now what was the targetns of the schema)

XML Schema Instance namespace

The schemaLocation attribute has two values. The first value is the namespace to use. The second value is the location of the XML schema to use for that namespace

How to reference XSD

- XSD can make use of namespaces!

  `schemaLocation` attribute in instance

  (value: pair namespaces-xsd)

- w/o targetNamespace:

  `noNamespaceSchemaLocation` attribute in instance

… sophisticated… don't bother too much for the moment. Find more in the references:

e.g.

http://ww.w3schools.com/

http://www.w3.org/XML/Schema

XML-Schema : Defining Elements

Elements

- Must have a name and a type (attributes)
- Either globally defined: as child of <schema>
- or locally defined: in context of other elements
  (can also have the same name as globally defined)
- Elements can refer to other (global) element or type definitions
- Elements can have an associated cardinality
- Complex or simple type:
  - simple: (refinements of) built-in types (e.g. strings, numbers, dates, etc.)
  - complex: can have nested elements, etc.

Example for a simple type element referring to a built-in type:

```
<xsd:element name = "employee" type = "xsd:string"/>
```

Example for a complex type element which does not refer to a global or built-in type declaration:

```
<xsd:element name = "marketing">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name = "employee" type = "xsd:string"
                   maxOccurs = "unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Element order in XSD

- In general, there is no preferred order within a schema document – you generally have to infer the "containing" element of a schema based upon which <xsd:element> references the highest elements in the tree.

  (unlike DTDs where you strictly define the document element)

XML-Schema : Defining Attributes

- Like elements, attributes must have a name and type
- Attributes can use custom data types (but **only simple types***!)
- Attributes can be restricted similar like in DTDs (`optional, fixed, required, default` values)

```
<xsd:attribute name = "country" type = "xsd:string" fixed="Austria"/>
```

```
<xsd:attribute name="partNum" type="ipo:SKU" use="required"/>
```

- Attributes can refer to other (global) attribute definitions:

```
<xsd:attribute ref="xml:lang" use="optional" />
```

* More on what exactly are simple types in the following slides….

XML-Schema : Annotations

- XML-schema provides several tags for annotating a schema: `documentation` (intended for human readers), `appInfo` (intended for applications) and `annotation`
- `Documentation` and `appInfo` usually appear as subelements of `annotation`

```
<xsd:annotation>
  <xsd:documentation xml:lang = "en">
    here goes the documentation text for the schema
  </xsd:documentation>
</xsd:annotation>
```

- *Remark: Different from normal comments in XML, since these annotations can be used in e.g. XML Schema editors, etc., difference is mainly conceptually*

XML-Schema : Data Type Definitions

XML-schema data types are either:

- Pre-built Simple Types
- Derived from Simple Types
- Complex Types

XML-Schema : Simple Types

- Simple types are elements that contain **data**
- Simple types **may not** contain attributes or sub-elements
- New simple types are defined by deriving them from built-in simple types

```
<xsd:simpleType name = "mySimpleDayOfMonth">
  <xsd:restriction base = "xsd:positiveInteger">
    <!--positiveInteger defines the minimum to be 1-->
    <xsd:maxInclusion value = "31"/>
  </xsd:restriction>
</xsd:simpleType>
```

XML-Schema : Some Built-in Simple Types

Simple Type	Example
xsd:string	Man, this day is long!
xsd:hexBinary	0FB7
xsd:integer	-12834, 0, 235
xsd:decimal	-1.23, 0, 5.256
xsd:boolean	TRUE, FALSE, 0, 1
xsd:date	1977-10-03
xsd:anyURI	http://www.Dot.com/my.html#a3

XML-Schema : Complex Types

- Complex types are elements that allow sub-elements and/or attributes
- Complex types are defined by listing the elements and/or attributes nested within
- Complex types are used if one wants to define sequences, groups or choices of elements

```
<xsd:complexType name = "myAdressType">
  <xsd:sequence>
    <xsd:element name = "Name" type = "xsd:string">
    <xsd:element name = "Email" type = "xsd:string">
    <xsd:element name = "Tel" type = "xsd:string">
  </xsd:sequence>
</xsd:complexType>
```

XML-Schema : Complex Types

In complex types elements can be combined using the following constructs:

- **Sequence:** all the named elements must appear in the sequence listed
- **Choice:** one and only one of the elements must appear
- **All:** all the named elements must appear, but in no specific order
- **Group:** collection of elements, usually used to refer to a common group of elements (only usable for global declarations and for references!), groups other sequences, choices or alls, for reuse.

XML-Schema : Complex Types

```
<xsd:complexType name = "myAdressType">
  <xsd:sequence>
    <xsd:element name = "Name" type = "xsd:string">
    <xsd:element name = "Email" type = "xsd:string">
    <xsd:element name = "Tel" type = "xsd:string">
  </xsd:sequence>
</xsd:complexType>
```

Cardinality can also be restricted (for choice and sequence!

```
<xsd:complexType name = "myAtMost2AdressType">
  <xsd:sequence minOccurs=0 maxOccurs=2>
    <xsd:element name = "Name" type = "xsd:string">
    <xsd:element name = "Email" type = "xsd:string">
    <xsd:element name = "Tel" type = "xsd:string">
  </xsd:sequence>
</xsd:complexType>
```

XML-Schema : Mixed, Empty and Any Content

- **Mixed content** is used if you want to model elements that includes both subelements and character data `<xs:complexType mixed="true">`
- **Empty content** is used to define elements that must not include any subelements and character data, implicit in the definition:

```
<xs:element name="product">
    <xs:complexType>
            <xs:attribute name="prodid" type="xs:positiveInteger"/>
     </xs:complexType>
</xs:element>
```

- **Any content** (the most basic data type) does not constrain the content in any way. The <any> element enables us to extend the XML document with elements not specified by the schema:

```
<xs:element name="person">
   <xs:complexType>
            <xs:sequence>
            <xs:element name="firstname" type="xs:string"/>
            <xs:element name="lastname" type="xs:string"/>
            <xs:any minOccurs="0"/>
   </xs:sequence>
   </xs:complexType>
</xs:element>
```

XML-Schema : Inheritance

- XML-Schema provides a „pseudo“ inheritance via type-derivations
- In XML-Schema all inheritance has to be defined explicitly
- New types can only be created by extending or restricting existing types
- Types can only be derived from one type – multiple inheritance is not supported

XML-Schema : Restricting a Type

- New simple types can be derived by constraining facets of a simple type
- The **XSD:RESTRICTION** element is used to state the base type

```
<xsd:simpleType name = "AustrianPostalCode">

  <xsd:restriction base = "xsd:integer">

    <xsd:minInclusive value = "1000">
    <xsd:maxInclusive value = "9999">

  </xsd:restriction>

</xsd:simpleType>
```

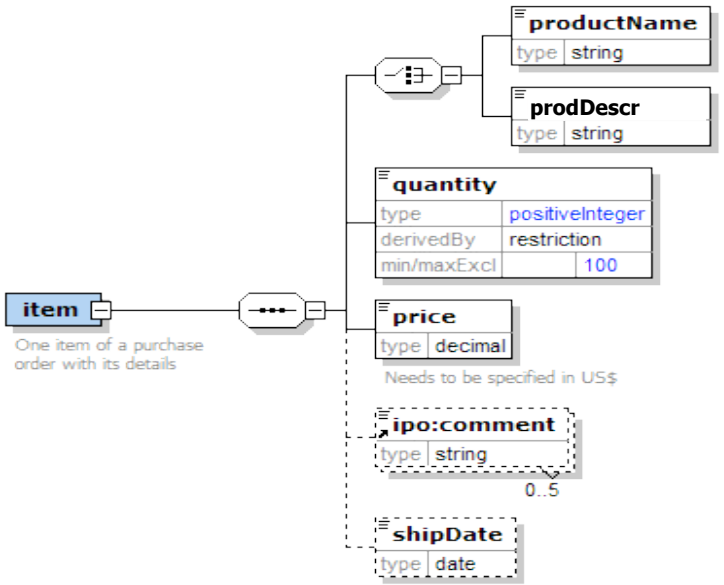
XML-Schema : Extending a Type

- New complex types can be derived by extending other types (simpleContent/complexContent)
- The **XSD:EXTENSION** element is used to state the base type

```
<xsd:complexType name = "myPrice">

  <xsd:simpleContent>

    <xsd:extension base = "xsd:decimal">
      <xsd:attribute name = "currency" type="xsd:string"/>
    </xsd:extension>

  </xsd:simpleContent>

</xsd:complexType>
```

When do you need this? E.g. if you want to attach attributes to a simple type element

Yet another complex example…




```
<element name="item">
  <annotation>
    <documentation>One item of a purchase order with its
                   details</documentation>
  </annotation>
  <complexType>
    <sequence>
      <choice>
        <element name="productName" type="string"/>
        <element name="productDescr" type="string"/>
      </choice>
      <element name="quantity">
        <simpleType>
          <restriction base="positiveInteger">
            <maxExclusive value="100"/>
          </restriction>
        </simpleType>
      </element>
      <element name="price" type="decimal">
        <annotation>
          <documentation>Needs to be specified in
                         US$</documentation>
        </annotation>
      </element>
      <element ref="ipo:comment" minOccurs="0" maxOccurs="5"/>
      <element name="shipDate" type="date" minOccurs="0"/>
    </sequence>
    <attribute name="partNum" type="ipo:Sku"/>
  </complexType>
</element>
```

Inheritance in XSD compared with "real" type systems/inheritance:

- Note that this inheritance is not FULL inheritance in the sense of the semantics defined for OOP or Ontologies, for instance there is no Bottom-up inheritance of instances:

  E.g. element employee is a subclass (restriction/extension) of complextype person, but in XSD no means to say that an employee is a person then…)

These were some but not ALL features of DTDs and XSD…

Some examples and exercices in the zip file I will pt on the lecture homepage might help!

Play around with it and find out more!

Overview

- XML-schema, a powerful alternative to DTDs
- XML APIs:
  - DOM, a data-object Model for XML
  - SAX, a simple API for XML
  - Validating Parsers based on DOM and SAX
- More XML Companion standards

XML Parsers&APIs: DOM vs. SAX

- Document Object Model (DOM)
  - DOM is a programming language independent object model and an API
  - The specification describes a defined XML-usage handling elements as objects for interaction with object-oriented programming languages such as Java (tutorials: Java DOM implementation of apache: Xerces)
  - DOM provides a **complete tree structure for all objects of XML-documents**
  - Not suitable for extremely large XML-files. Good for forms/editors, simple to program with.
  - DOM is a (bunch of) W3C Standard recommendation(s).

- Simple API for XML (SAX)
  - More "low-level" API for XML application processing with the help of object-oriented programming languages such as Java. (tutorials: SAX implementation of apache: Xerces)
  - **Event-based** rather than tree-based, not the whole tree in memory, sequential.
  - SAX delivers an XML-element to an output stream and is suitable for processing large XML-Files. Good for App2App exchange.

- An *XML parser* reads an XML document and provides it in a form of DOM or in its own structure with SAX-events for further processing. A *validating parser* also checks with DTD/XSD.

What is DOM ?

- DOM(Document Object Model)
  - Was developed by W3C
  - Specify how future Web browser and embedded scripts should access HTML and XML documents

This and the following slides are very much borrowed from:
http://oopsla.snu.ac.kr/research/object/open_xml/8_xmldomsax_noted.ppt

Java implementation

- SUN provides a class for parsing XML, called Xml Document.
- Xml Document methods to parse XML file, build the document tree.
- To use the SUN parser

```
=> import org.w3c.dom.*;
   import com.sun.xml .tree.*;
   import org.xml.sax.*;
```

Check docs for these to learn details!

Nodes (1/4)

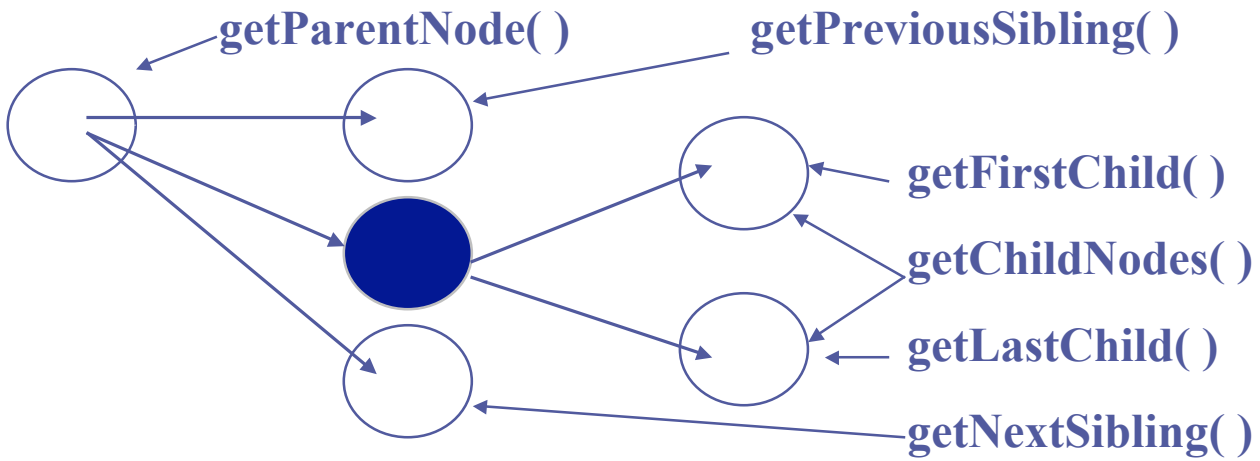
- Nodes
  - describe elements, text, comments, processing instructions, CDATA section, entity references ...
- The Node interface itself defines a number of methods.
  1. Each node has characteristics (type, name, value)
  2. Having a contextual location in the document tree.
  3. Capability to modify its contents.

Nodes (2/4)

- Node characteristics
  - getNodeType => determining its type
  - getNodeName => returning the name of the node
  - setNodeValue => replacing the value of node
  - hasChildNodes => whether node has children or not
  - getAttributes => accessing attribute

Nodes (3/4)

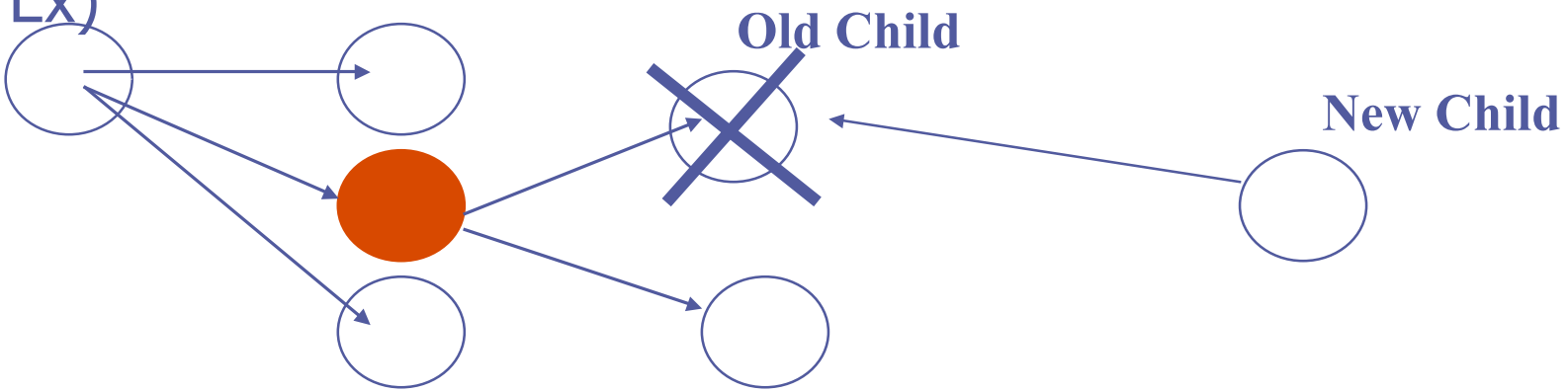
- Node navigation
  - When processing a document via the DOM interface, it is to use node as a stepping-stones.
  - Each node has methods that return references to surrounding nodes.



Nodes (4/4)

- Node manipulation
  - remove child method.
  - appendChild method
  - insertbefore method
  - replaceChild method

Ex)



Documents

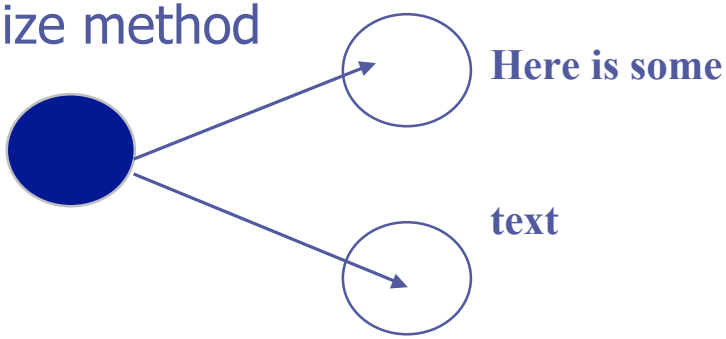
- An entire XML document is represented by a special type of node.
  - getDoctype
  - getImplementation
  - getDocumentElement
  - getElementsByTagName

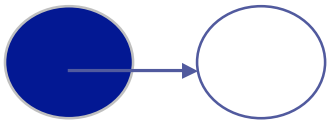
Elements

- Element interface
  - Extends the Node interfaces
  - Adds element-specific functionality
  - General element processing
    - getTagName method
    - getElementsByTagName method
    - normalize method

Example

normalize:





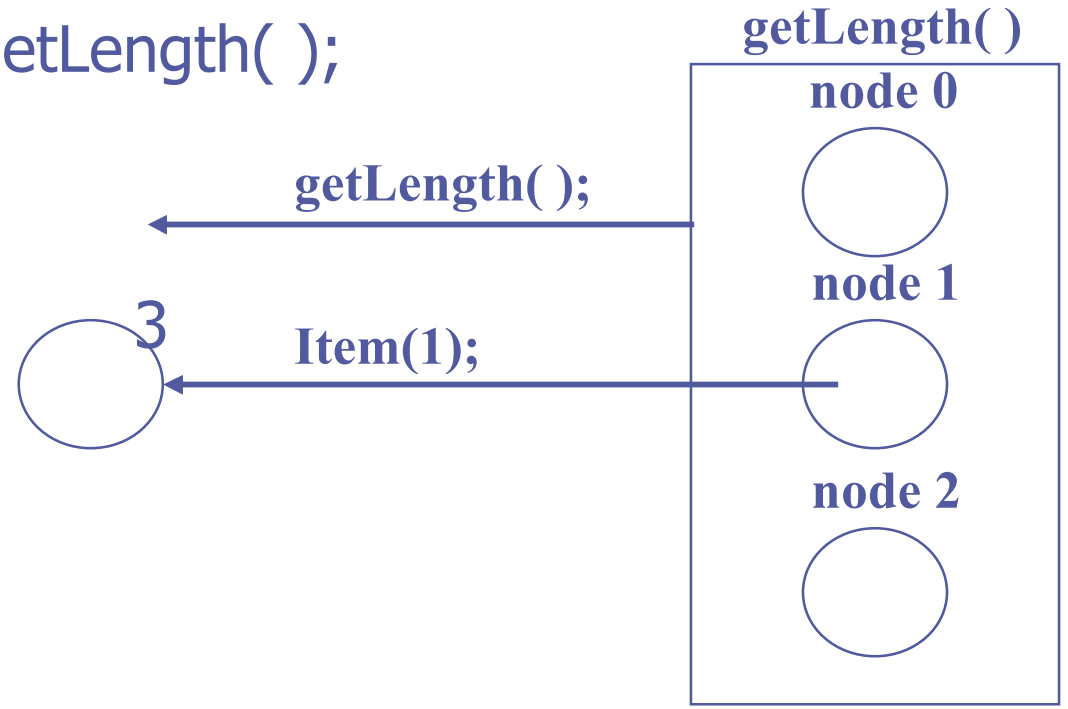
“Here is sometext”

Attributes

- Attribute characteristics
  - getName method
  - getValue
  - setValue
  - getSpecified
- Creating attribute
  - createAttribute

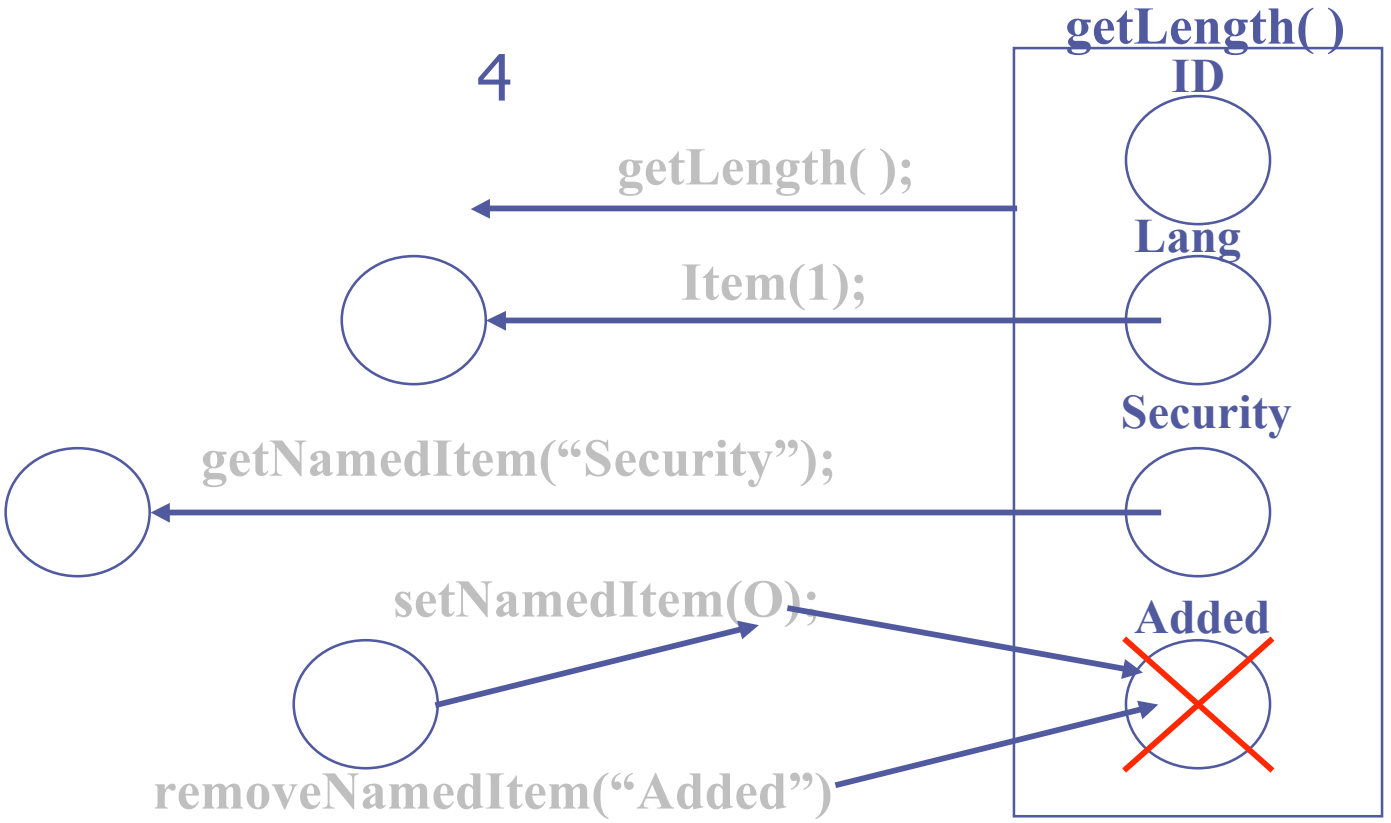
Node lists

- The **Nodelist** interface contains two method
  - Node item(int index);
    int getLength( );



Named node maps

- The **NamedNodemap** interface is designed to contain nodes, in no particular order, that can be accessed by name.



What is SAX?

- **SAX(the Simple API for XML)**
  - Is a standard API for **event-driven** processing of XML data
  - Allowing parsers to deliver information to applications in digestible chunks

Call-backs and interfaces

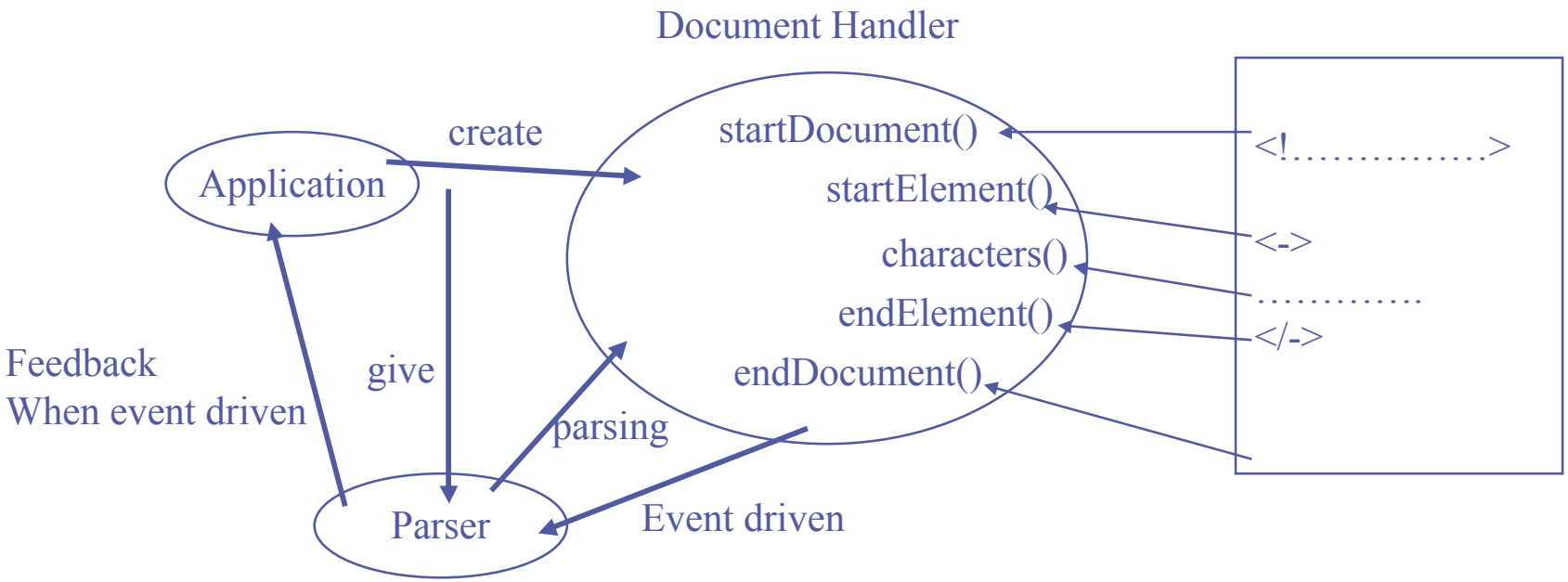
- The SAX interface are:
  - Parser
  - Document Handler
  - AttributeList
  - ErrorHandler
  - EntityResolver
  - Locator
  - DTD Handler

The Parser

- The Work of Parser
  - The parser developer creates a class that actually parses the XML document or data stream
  - The parser reads the XML source data
  - Stops reading when encounters a meaningful object
  - Sends the information to the main application by calling an appropriate method
  - Waits for this method to return before continuing

Document handlers

- In order for the application to receive basic markup events from the parser, the application developer must create a class that implements the DocumentHandler interface.



Attribute lists

- A wrapper object for all attribute details
  - int getLength(); … to associate how many attributes are present.
  - String getName(int i); … to discover the name of one of the attributes
  - String getType(int i); … when a DTD is in use, to get a data type
  - String getType(String name); assigned to each attribute.
  - String getValue(int i); … to get the value of an attribute
  - String getValue(String name);

Error handlers

- When the application needs to be informed of warnings and errors
  - It can implement ErrorHandler interface

Locators

- Necessity
  - An error message is not particularly helpful when no indication is given as to where the error occurred.
- Locator interface
  - can tell the entity, line number and character number of the warning or error

Handler bases

- HandlerBase class
  - Providing some sensible default behavior for each event, which could be subclassed to add application-specific functionality
  - E.g. DTDHandler

XML Application Architecture

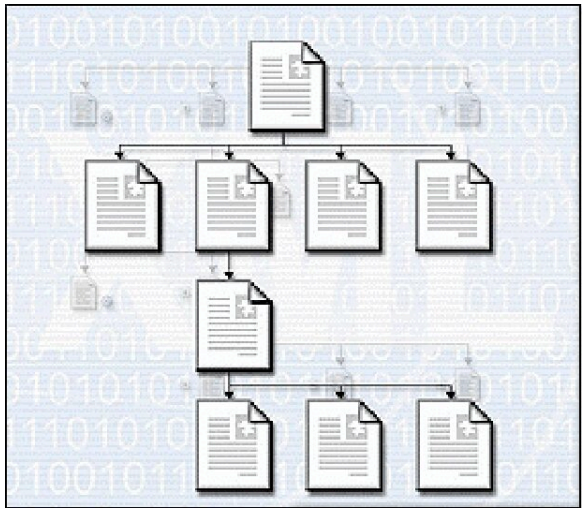
- An XML application is typically built around an XML parser
- It has an interface to its users, and an interface to some sort of back-end data store

Kinds of Parsers

- Validating versus non-validating parsers
  - Validating parsers validate XML documents as they parse them with respect to a DTD or an XSD
  - Non-validating parsers ignore any validation errors
- Parsers that support the Document Object Model(DOM)
- Parsers that support the Simple API for XML(SAX)

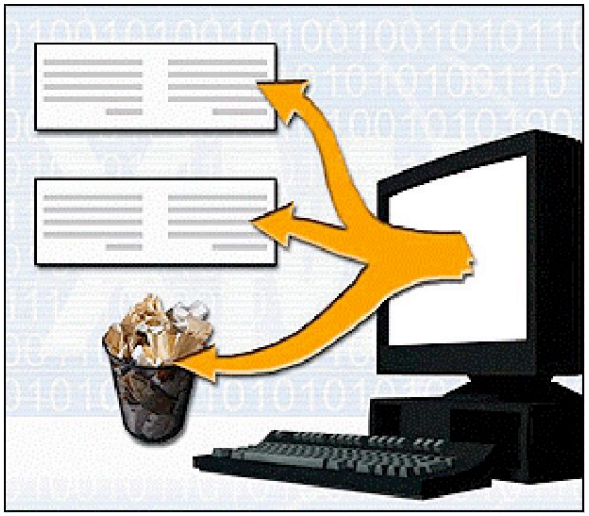
DOM Parser

- Tree structure that contains all of the elements of a document
- Provides a variety of functions to examine the contents and structure of the document

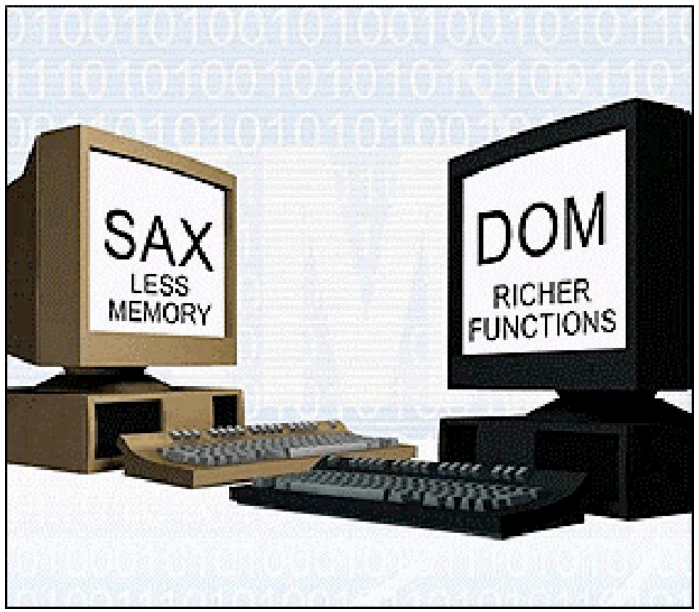


SAX Parser

- Generates events at various points in the document
- It's up to you to decide what to do with each of those events



DOM vs SAX



- Why use DOM?
  - Need to know a lot about the structure of a document
  - Need to move parts of the document around
  - Need to use the information in the document more than once
- Why use SAX?
  - Only need to extract a few elements from an XML document

A sample XML parser implementation:

- Xerces: xml.apache.org/

  provides full implementation of SAX and DOM (level 2) APIs for JAVA, C++, etc.

Overview

- XML-schema, a powerful alternative to DTDs
- XML APIs:
  - DOM, a data-object Model for XML
  - SAX, a simple API for XML
  - Validating Parsers based on DOM and SAX
- More XML Companion standards

XML : XPath

- XPath is a non-XML language used to identify particular parts of XML documents.
- XPath lets you write expressions that refer to the document's first person element, the seventh child element of the third person element, the ID attribute of the first person element whose contents are the string "Fred Jones“,…
- XSLT and XPointer use XPath to identify particular points in an XML document.

XML : XPointer

- XPointer defines an addressing scheme for individual parts of an XML document.
- XPointers enable you to target a given element by number, name, type, or relation, to other elements in the document.
- XPointers uses the XPath syntax to identify the parts of the document they point to.

XML : XML Linking Language

- The XML Linking Language (XLink) allows elements to be inserted into XML documents in order to create and describe links between resources.
- Generalizes Link Concept of HTML, based on XPointer.
- XLink provides a framework for creating both basic unidirectional links and more complex linking structures, e.g. linking relationships among more than two resources, associate metadata with a link,...

XML : XML Style Language

- XSL (XML Style Language) is a language for expressing stylesheets for XML documents
- XSL describes how to display an XML document of a given type
- XSL defines a set of elements called **Formatting Objects** and attributes (in part borrowed from CSS2 properties and adding more complex ones)

XML : XSLT

- XSLT (XSL Transformations) is a language for transforming XML documents into other XML documents.
- XSLT is also designed to be used independently of XSL. However, XSLT is not intended as a completely general-purpose XML transformation language. Rather it is designed primarily for the kinds of transformations that are needed when XSLT is used as part of XSL.
- A transformation expressed in XSLT describes rules for transforming a source tree into a result tree.

Important for Data Integration/Mediation!

Etc. etc.

References

- **XML-Schema :**
  http://www.w3.org/XML/Schema
  (also has links to useful tools, etc.)
  Fallside, D. *XML Schema Primer*,
  http://www.w3.org/TR/xmlschema-0/

- **DOM & SAX:**
  Some very nice introduction to DOM&SAX:
  http://oopsla.snu.ac.kr/research/object/open_xml/8_xmldomsax_noted.ppt
  Xerces: http://xml.apache.org

Examples will be provided on the Lecture-homepage (zip file)!