Network Services, VU 2.0

XML Technologies / 1

Dipl.-Ing. Johann Oberleiter
Institute for Informationsystems, Distributed Systems Group

Agenda

- SGML & XML & HTML
- XML Schema & DTD
- XPath
- XSLT & XSL

SGML

- Standard Generalized Markup Language
  - Initial goal to represent text in electronic form
  - Device & System Independent
- Meta-Language
  - Means for formally describing a language
    = Markup Language
  - Powerful
  - Very complex
- Separation of Content, Structure and style
- Logical ancestor of HTML, XML
- Used in Publishing industry
  - Continously replaced by XML

XML

- eXtended Markup Language
  - Initial goal to represent data in electronic form
  - Device & System independent
- Meta-Language
  - Markup language
  - Less complex than SGML
  - Powerful
  - May be parsed by SGML parsers with special extensions
- Base for almost all new data representation languages
- Strong backup in Software Industry
  - Microsoft
  - IBM

HTML

- No Description in this lecture (!)
- http://de.selfhtml.org/
- http://www.w3.org/TR/REC-html40/

- CSS will be explained
- Every technician should be aware of HTML
  - Especially computer scientists

Motivation for XML

- Problems with HTML
  - Intended for visualization
  - Mixes content and style (layout)
  - Difficult to automatically transform
- XML
  - Describes information in a document
  - No visualization
  - Says what a document means

More HTML problems

- HTML is static
  - Not extensible
  - Set of elements is fixed
- No Semantic information
- Not designed for device-independence
  - Different on desktop browsers, PDAs, ...
- Layouting features rather weak
  - CSS

XML / 1

- Meta-language
  - Defining new languages
  - Example: XHTML
    - Redesigned HTML, conforms to XML
- Application of XML
  - Introducing such a language
- Supports structure
  - Through structure of tags
- Supports semantics
  - Meaning of tags
    - <Person>Mustermann</Person> vs. "Mustermann"
  - Important for automation

XML / 2

- Supports Reuse
  - Reuse of Structure (DTD, Schemas)
    - Tree structures
  - Reuse of Content (XSLT)
    - Transformation to other representations
- Supports Validation
  - Checking if XML conforms to parrticular DTD or Schema
- Support for encodings
  - Unicode -> supports almost all characters
  - Standardized way to say which character set is used

XML / Example 1

```
<Person>
    <Nachname>Mustermann</Nachname>
    <Vorname>Vorname</Vorname>
    <Addresse>
        <Strasse>Argentinierstrasse 8</Strasse>
        <Ort>Wien</Ort>
        <PLZ>1040</PLZ
    </Addresse>
</Person>
```

XML / Example 2

- Whitespaces don't matter:

```
<Person>
<Nachname>Mustermann</Nachname><Vo
   rname>Vorname</Vorname><Addresse
        <Strasse>Argentinierstrasse
   8</Strasse><Ort>Wien</Ort><PLZ>1040<
   /PLZ</Addresse></Person>
```

Goals for XML

- Easy to read and process
  - More important: easy for machines
- Separation of layout and content
- Typed documents
- Compatible with SGML
- Unicode

Application Areas / 1

- World Wide Web
  - XML sent to client, rendering on client
  - XML rendered on server, HTML sent to client
- Separation of layout and content
- Automatic generation of navigational structures

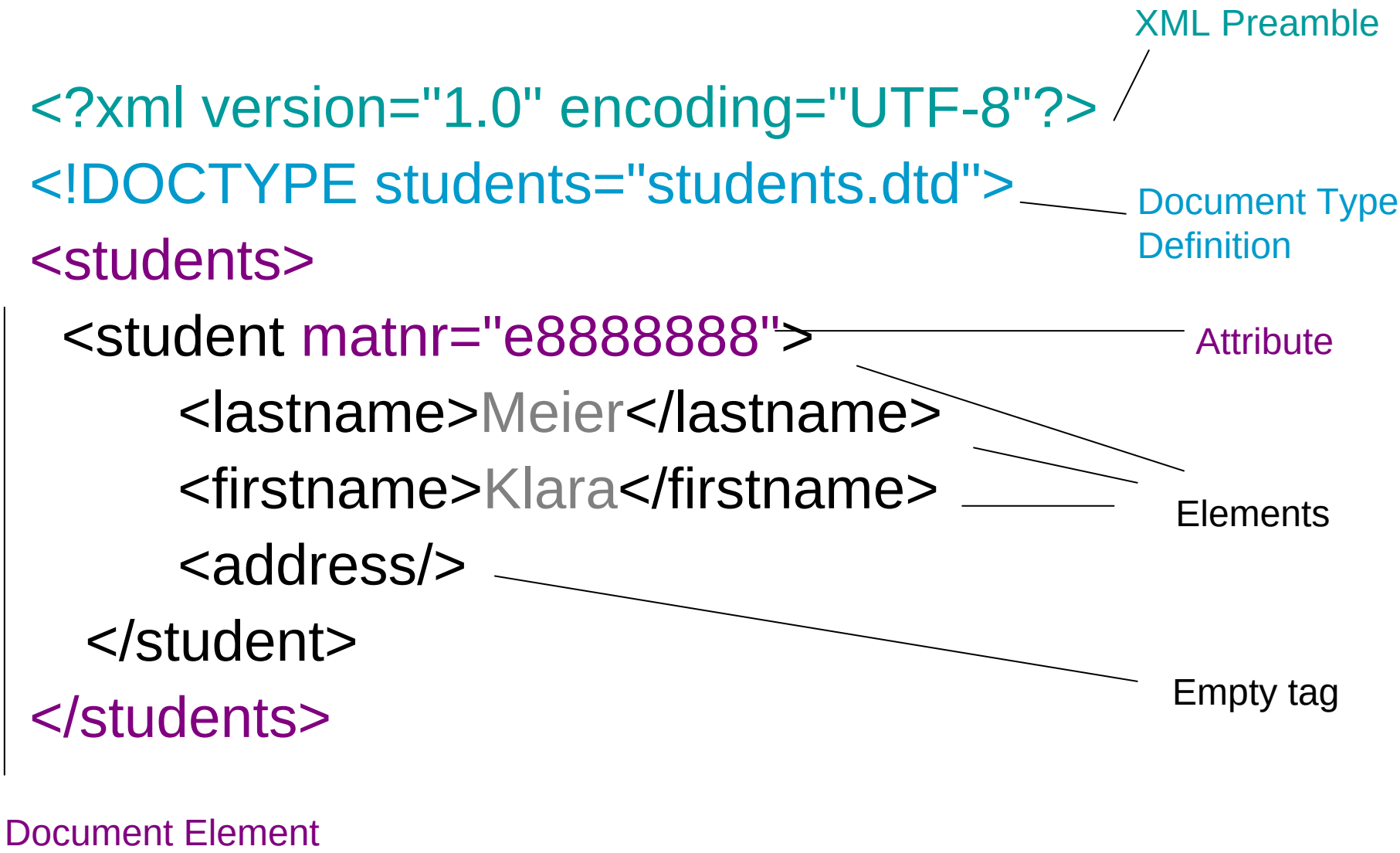
Application Areas / 2

- Data Exchange / Interoperability
  - SOAP (later)
  - WebDAV (later)
  - BPEL (Business Process Execution Language)
- XML to enhance databases
  - Most commercial DBs support XML as result-set
  - Next generation:
    - Support XML as first class datatype
    - Supports querying within XML structures
- XML as structured databases
  - Eg. Apache Xindice

Application Areas / 3

- Domain Specific Languages (DSL)
  - MathML, SVG, MusicML, RDF, XMI
  - Ant build.xml
  - .NET Configuration files

XML Structure



XML Parts

- XML Preamble
  - Not required, highly recommended
  - "1.0" fixed version
  - Encoding: US-ASCII, UTF-8, ISO-8859-,1 UTF-16
  - Standalone: yes/no
- Document Type Definition
  - defines structure of XML file (=XML Infoset)
  - Defines root element name
  - Only required for valid documents
- Document Element
  - Root of XML tree
  - At the same level as Comments and processing instructions
- Processing Instructions
  - At same level as XML Preamble
  - <?mso-application progid="Excel.Sheet"?>
  - Special meaning for some programs

XML Infoset

- Elements
  - Structuring facility, can be nested
  - Opening and closing tag
  - Empty tags (closed)
  - Content Models
    - Elements only
    - Mixed (text & child element)
- Attribute
  - Information bundled within attributes (name-value)
  - Multiple attributes
  - Never nested
- Text
  - Strings & characters in encoding format
  - Meta character need to be escaped
    - < > & ' &quote
- Comments
  - <!- - an XML comment - ->

Well-Formedness

- Minimal Requirements for "good" XML document
  - For all start tags exist end tag
  - Exactly one document element
  - Correct cascading of elements
  - Only comments and PIs out of document element
  - All attributes in quotes
  - No double attributes in one element

XML Parsing APIs

- XML parser support everywhere
  - Check well-formedness of XML files
  - Java, .NET, C, Perl, ...
  - Automatic generation of programming language structures
    - Java JXB
    - .NET XML
  - Otherwise resolving structure manually with APIs
- DOM (Document Object Model)
  - tree-representation of XML
- SAX (Simple API for XML)
  - Event based handling

SAX (Simple API for XML)

- Event based processing
- Callbacks
  - content, errors, ...
- Event fired
  - every time start of new tag encounterd
  - every time end of tag encountered
- Not necessary to load whole XML file in memory
  - Fast
  - Serialization (import/export)
  - No possibility of accessing already parsed elements
  - Read-Only (SAX events are readonly notifications)
- Application scenarios
  - Only certain aspects (events can be ignored)
  - Server-side (Performance)

DOM (Document Object Model)

- Interface to tree-representation of XML document
- Language independent
- Mirrors complete data structure
  - Easy to go back
- Slow compared to SAX

DTD / Schema

- Valid XML documents
  - Well-Formed & conforms to rules in DTD or Schema
  - An application may required a certain structure
- Meta-Information about documents
  - DTD / Schema describe a set of documents (that conform to the rules)
- Parsers and representation classes can be generated from DTDs / Schemas

DTD (Document Type Definition)

- Written in its own language
- Rules
  - Which elements may be used
  - Which content models they have
    - element, text, empty, mixed, any
  - How may elements be nested
  - Which Attributes are allowed
- External vs. Internal
  - If DTD is external to XML document
- Public identifier
  - Public identifier for standard document classes suitable
  - Can only be external
- System identifier
  - For everything else

DTD Element Definition

- <!ELEMENT *myelement* contents>
  - Declares Structure of an element myelement
  - Number of occurences (?,+,*)
  - Grouping "()", Sequences ",", Choices "|"
  - Text #PCDATA, ANY, EMPTY
- <!ATTLIST >
  - Declares Occurence of attributes (#REQUIRED,#IMPLIED,#FIXED)
  - Attribute types
    - CDATA (text),
    - IDs (unique ids) , IDREF,IDREFS references ids
    - ENTITY, ENTITIES references to an entity
    - NMTOKEN, NMTOKENS names according to XML spec
- <!ENTITY entity-name "my-entity">
  - Kind of Macro, entity value will be replaced during parsing of XML
  - Refered within XML with &my-entity;
  - Predefined entities: < > & " '
  - Every character may be represented as character entity,
    - eg. ¶ &xAB;

DTD Sample

```
<!ELEMENT students(student+)>
<!ELEMENT student(lastname,firstname,adress)>
<!ATTLIST student matnr CDATA #required>
<!ELEMENT lastname(#PCDATA)>
<!ELEMENT firstname(#PCDATA)>
<!ELEMENT adress(#PCDATA)>
<!ENTITY city "Vienna">
```

```
<students>
  <student matnr="e8888888">
            <lastname>Meier</lastname>
            <firstname>Klara</firstname>
            <adress>&city;</adress>
    </student>
</students>
```

Problems of DTD

- No datatypes
  - 4 generic content models
  - Fixed set of attribute types
- No reuse
- No extension mechanism
  - No user defined types
- No namespaces
  - Nameclashes when different XML files are combined
- No import of other DTDs

XML Schema / 1

- Successor of DTD
- Formulated in XML
- Context-free regular grammar for defining arbitrary XML structures
- Better support for versions of elements and attributes
  - More restrictions, more checks
- No support for Entities!

XML Schema / 2

- No "internal" schemas
- Linked to document with
  - schemaLocation attribute
  - noNamespaceSchemaLocation
- Global definition of elements
  - Children of document element
  - Name+type+option attributes
- Local definition of elements
  - Context of other elements
  - Name+type+option attributes
- Element references
  - References global elements
  - Supports Reuse of element definitions

XML Schema / Builtin Datatypes

- Primitive Datatypes
  - string,boolean,decimal,float,double,date,base 64Binary,duration,...
  - atomic
- Derived Datatypes
  - Subtypes of primitive types
  - integer,long,int,short,byte
  - negativeInteger, positiveInteger, ...
  - ID/IDREFS, ENTITY,ENTITIES*

XML Schema / Sample

```
<schema
   xmlns=http://www.w3.org/2001/XMLSchema>
   <element name="global" type="string">
        <element name="local" type="float"/>
        <element ref="global2"/>
   </element>
   <element name="global2" type="string"/>
</schema>
```

XML Schema / Simple vs Complex Type

- Simple Types
  - Used to refine built-in or derived types
  - Eg. "string with a fixed length of 4 characters"
- Complex Types
  - Mechanisms for nesting
  - Support Attributes

XML Schema / Sample 2

```
<student matnr="e8888888">
    <lastname>Meier</lastname>
    <firstname>Klara</firstname>
    <adress>Vienna</adress>
</student>
```

```
<element name="student">
    <complexType>
        <sequence>
            <element name="lastname" type="string"/>
            <element name="firstname" type="string"/>
            <element name="address" type="string"/>
        </sequence>
    </complexType>
</element>
```

XML Schema / Element

- Definition of Type
  - Either anonyous <complexType> child
  - Reference to named Type via attribute
- Element Modifiers
  - Name: unqualified (without namespace)
  - Type
  - ID: unique ID
  - maxOccurs: max number of occurences
  - minOccurs
  - Ref: to another element declaration
  - Default: default value
  - Nillable: optionally there may be no content

XML Schema / Sample 3

```
<sequence>
    <element name="lastname" type="string"      maxOccurs="40"/>
    <element name="firstname" type="string"     maxOccurs="20"/>
    <element name="address" type="string"       minOccurs="0"
                                                maxOccurs="unbounded"/>
</sequence>
```

XML Schema – Selection Models

- <sequence>
  - All Elements of a sequence
- <choice>
  - One out of many
  - Better than ? Because of minOccurs/maxOccurs
- <all>
  - All elements in any order

XML Schema – Simple Types

- Inherit from built-in types
  - primitive or derived types
- No children
- Used to refine base type
  - Only Restriction of a base type
  - Refinement with facets
    - length,minLength,maxLength,minInclusive,...
    - pattern (regular expressions), enumeration
    - totalDigits, fractionDigits
    - whitespace (use of whitespace in a type)

XML Schema – Simple Types

```
<simpleType name="matnr">
  <restriction base="string">
    <pattern value="e\p{Nd}{8}</pattern>
  </restriction>
</simpleType>
```

XML Schema – Complex Types

- Inheritance by restriction
  - Has to redefine all elements that are taken from base type
- Inheritance by extension
  - Add a new feature to a type
  - Like (code) inheritance in OO languages (subclassing)
- Inheritance by redefinition
  - Affects all occurences of a type
- Abstract elements & types

XML Schema – Content Models

- Simple Content
  - Only character content
  - No elements
- Complex Content
  - Elements only or mixed content
- Attributes possible with both content models
  - only with complex types(!)

XML Schema / Sample

```
<complexType name="Base">
   <sequence>
      <element name="Firstname" type="string"/>
      <element name="Lastname" type="string"/>
   </sequence>
</complexType>
```

XML Schema / Inheritance by Restriction

```
<complexType name="RestrictedType>
    <complexContent>
        <restriction base="Base">
        <sequence>
            <element name="Lastname">
                <simpleType>
                    <restriction base="string">
                        <maxLength value="20"/>
                    </restriction>
                </simpleType>
            </element>
        </sequence>
        </restriction>
    </complexContent>
</complexType>
```

Redefined type of Lastname (restricted type)

Element Firstname removed

XML Schema / Inheritance by extension

```
<complexType name="ExtType">
   <complexContent>
      <extension base="Base">
         <sequence>
            <element name="Phone" type="string"/>
         </sequence>
      </extension>
   </complexContent>
</complexType>
```

XML Schema – Substitution Groups

- A set of elements that can be used interchangeably for another element

- <element name="color" type="string">
- <element name="farbe" substitutionGroup="color" type="string">

XML Schema – Substitution Groups Sample

```
<car>
   <color>red</color>
</car>
```

```
<car>
   <farbe>blau</farbe>
</car>
```

XML Schema Annotations

- Additional documentation
  - For humans (with <documentation> tag)
  - For applications (with <appinfo> tag)
    - Interesting for automatic tools
- Placed
  - Before and after global components
  - Only at beginning of local components

XML Namespaces

- Avoid name clashes when documents are merged or interchanged
  - Unique naming
  - <Address> element of two different origins do not have necessarily the same structure
  - Otherwise complete XML file (or schema) has to be parsed
- Prefix + Unique identifier
  - Prefix is abbreviation for unique identifier
  - Unique identifier is usually a URL
- Used namespaces are declared in document element

XML Schema - Sample

```
<xyz:rootDoc xmlns:xyz="http://www.xyz.com">
        <xyz:LastName>Ostbahn</xyz:LastName>
</xyz:rootDoc>
```

XML Schema - Namespace

- Namespace of XML Schema http://www.w3.org/2001/XMLSchema
  - Usual prefix xsd
  - Contains all XML Schema XML elements
- Target Namespace
  - when element of a schema is used in instance document
  - All elements declared in schema have to use this namespace

XML Schema - Sample

```
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xsd:targetNamespace="http://www.xyz.com"/>
        <xsd:element name="Lastname"
        type="xsd:string"/>
        ...
</xsd:schema>
```

```
<xyz:rootDoc xmlns:xyz="http://www.xyz.com">
        <xyz:LastName>Ostbahn</xyz:LastName>
</xyz:rootDoc>
```

XSL

- eXtended Stylesheet Language
- Consists of
  - XSL Transformations
  - XML Path Language
  - XSL Formatting Objects (XSL-FO)

XPath

- Selection and addressing language
  - for XML (of course)
  - Based on XML's tree structure
- XPath expressions
  - Select single nodes or nodesets (collection of nodes)
- Evaluation always based on local node (context)

XPath - Example

```
<students>
    <student matnr="7523333">
        <lastname>Gates</lastname></student>
    <student matnr="8524234">
        <lastname>Thorwalds</lastname></student>
</students>
```

/
/student
/student/lastname
//lastname
/student/*/lastname
/student/*/lastname/../
/student[@matnr='7523333']/lastname

XPath - Axes

- Navigation within XML tree with axes
  - child, parent (abbreviation ..), self (.)
  - ancestor, ancestor-or-self (parent)
  - descendant,descendant-or-self (children)
  - following, following-sibling (sequence)
  - preceding, preceding-sibling (sequence)
- Within XPath: [axis-name]::[node-name]
  - /student/child::lastname = /student/lastname
- attributes axis (@)
- namespace axis

XPath – testing with predicates

- /student/[predicate]/lastname
- Multiple predicates in one expression
  - /student[@matnr='7523333']/name[@nametype='first']
- Attribute testing by value good
- Element testing difficult
  - because of whitespaces

XPath – Selecting other nodes

- Text Nodes
  - text()
  - Eg. /student/[@matnr='7523333']/lastname/text()
- Any node
  - node()
  - /student/* <> /student/node()
  - Difference: node() selects any node, * selects only element nodes

XPath – Expression Types

- Node sets
  - All Node selecting expressions
- Boolean
- Numbers
- Strings
- Result tree fragments
  - Portion of XML document not complete node or node set
  - May be converted to string

XPath - Functions

- For Node-sets
  - position() returns number of node in node-set
    - eg. /student[position() = 2]
  - last()
  - count(node-set)
    - eg. count(//students)
  - name(node-set)
    - Name of first node in node set
    - local-name, namespace-uri

XPath – Boolean & Numbers

- Boolean values
  - Predefined: true & false
  - Results of relational operators
    - =,!=,<,>,<=,>=
    - ! Use < instead of <
  - Operators and, or, not()
- Numbers
  - Expressions implicitly converted to a number
  - Arithmetic operators
    - +, -, *, div, mod
  - Functions: floor(), ceiling(), round(), sum()

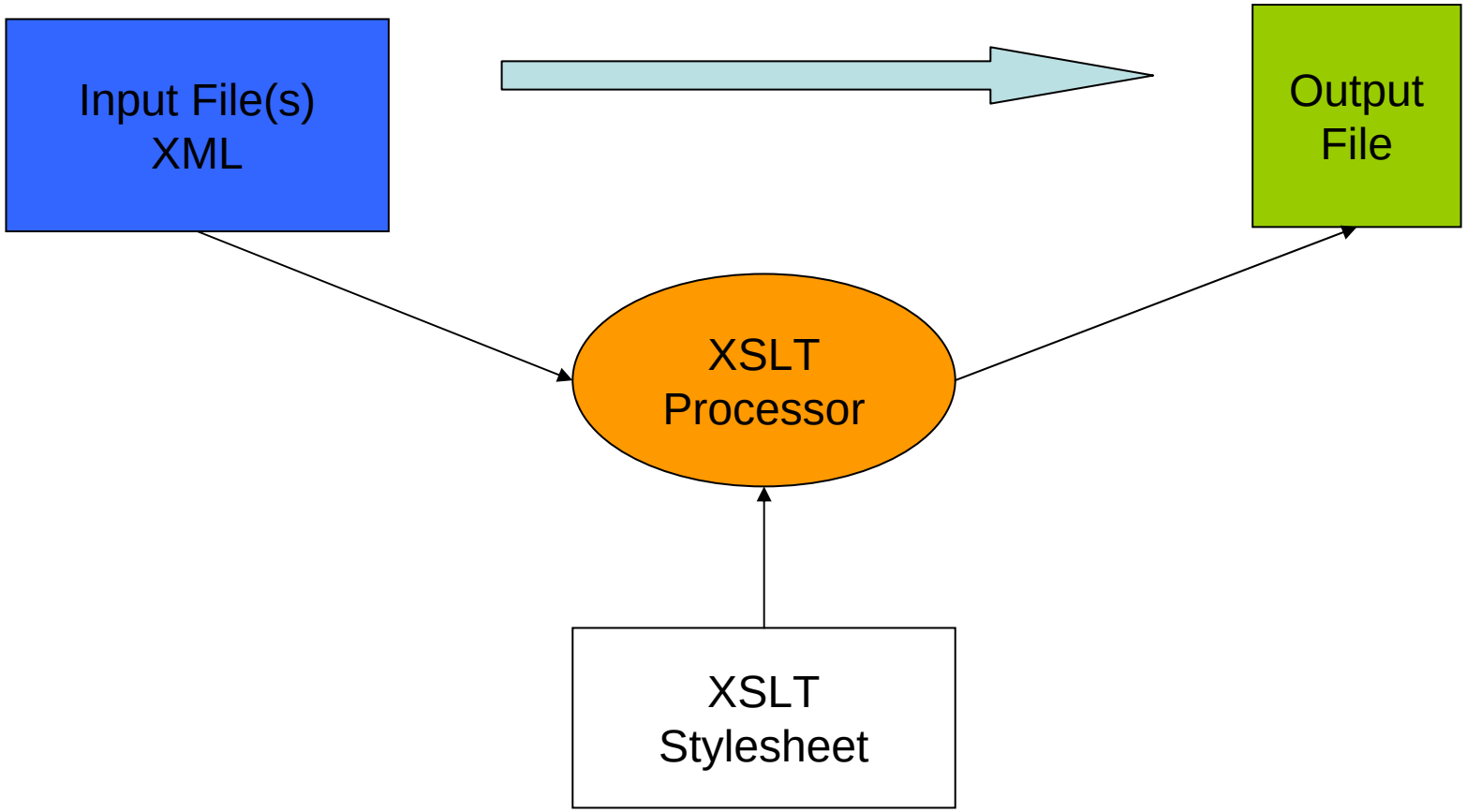
XPath – String

- Functions on string
  - starts-with(s, prefix)
  - contains(s, substring)
  - substring(s, offset, length)
  - normalize-space(s)
  - string-length
  - concat(s1,s2)
  - format-number(number, format-string)
  - ...

XSL Transformations

- Transformation language
  - XML language
- Input is XML
- Output may be
  - XML
  - Text
  - HTML
  - Other formats via extensions
- Rule based

XSLT Transformation



XSLT – Basic principles

- Transformation rule

<xsl:template match="[XPath-Expression]">

  Substitution-Part

</xsl:template>

When XPath-Expression evaluates to true for a node the substitution part is applied and allows modification of the tested node.

XSLT – Elements for Substitution

- <xsl:value-of select="xpath-expr">
  - Inserts the text value of an XPath expression into the output

- <xsl:template match="//student">
  <xsl:value-of select="lastname"/>
- </xsl:template>

XSLT – Elements for Substitution

- <xsl:apply-templates select="xpath-expr">
  - Specifies where processing shall continue
  - Searches for template rules in select attribute
  - If select omitted processing is done for all elements

- <xsl:text>
  - Outputs normal text
- <xsl:element>, <xsl:attribute>
  - Outputs an element or an attribute
  - Only useful for XML-like output

XSLT - Sample

```
<xsl:template match="/">
   <xsl:apply-templates select="student">
</xsl:template>

<xsl:template match="student">
   <xsl:text>Student:</xsl:text>
   <xsl:value-of select="lastname/text()"/>
</xsl:template>
```

XSLT – Default Rules

- Normally each node requires a rule
  - Otherwise processing stops
  - Tedious to write a node for all elements
- Solution: Default Rules
  - <xsl:template match="*|/">
    - <xsl:apply-templates/>
  - </xsl:template>

XSLT Sample – Generate HTML

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl=http://www.w3.org/1999/XSL/Transform>
    <xsl:output method="html"/>
    <html>
        <head>...</head>
        <body>
            <ul>
                <xsl:apply-templates select="//student"/>
            </ul>
        </body>
    </html>

    <xsl:template match="student">
        <lil><xsl:value-of select="lastname/text()"/></li>
    </xsl:template>
</xsl:stylesheet>
```

XSLT Sample – Resulting HTML

- Gates
- Thorwalds

XSLT Choices

- <xsl:if test="xpath-expr">
  - Supports conditional processing based on an expression
  - No else (!)

- <xsl:choose>
  - Switch-like statements switch in Java
  - Single cases in <xsl:when test="xpe"> elements
  - With <xsl:otherwise> else clause possible

XSLT – Iteration / 1

- <xsl:for-each>
  - Iterates over a node-set
- Example
  - <xsl:template match="/">
    - <xsl:for-each select="student">
      - <xsl:value-of select="lastname"/>
    - </xsl:for-each>
  - </xsl:template>
- What's the difference to <xsl:apply-templates>?

XSLT – Sample XML file

```
<student>
   <name>Bill</name>
   <note lvaname="EProg">4</note>
   <note lvaname="AlgoDat">2</note>
   <note lvaname="Math 1">1</note>
</student>
```

Want following text:

Noten von Bill EProg 4, AlgoDat 2, Math 1

XSLT – Iteration / 2

- Solution with <xsl:apply-templates>

```
<xsl:template match="student">
      <xsl:text>Noten von </xsl:text><xsl:value-of
         select="name/text()"/><xsl:apply-templates select="note"/>
</xsl:template>

<xsl:template match="note">
    <xsl:value-of select="@lvaname"><xsl:text> </xsl:text><xsl:value-of
    select="./text()"/><xsl:text>,</xsl:text>
</xsl:template>
```

Problem: too many commas, will result in

Noten von Bill EProg 4, AlgoDat 2, Math 1,

XSLT – Iteration / 2

```
<xsl:template match="/">
   <xsl:for-each select="student">
         <xsl:text>Noten von </xsl:text>
         <xsl:value-of select="name/text()"/>
         <xsl:value-of select="@lvaname"/>
         <xsl:value-of select="./text()"/>
         <xsl:if test="not last()">
                  <xsl:text>,</xsl:text>
         </xsl:if>
   </xsl:for-each>
</xsl:template>
```

Noten von Bill EProg 4, AlgoDat 2, Math 1

XSLT – Variables

- Can define Constants
  - <xsl:variable name="myvar">Mein eigener Text</xsl:variable>
- Can contain XPath expressions
  - <xsl:variable name="myvar" select="xpath-expr"/>
- Cannot be modified !
- Globally as children of document element
- Locally within a template rule
- Referenced using $name
  - <xsl:value-of select="$myvar"/>

XSLT - Modes

- Different processing rules for the same element
  - Supports different "states" of a document
  - Supports inserting the same contents twice within a document
- Name of mode attached
  - To rule template
  - Specified in apply-templates

XSLT – Modes / 2

```
<xsl:template match="/">
    <xsl:apply-templates select="//student" mode="global"/>
    <list>
        <xsl:apply-templates select="//student" mode="list"/>
    </list>
</xsl:template>

<xsl:template match="student" mode="global">
...
</xsl:template>

<xsl:template match="student" mode="list">
...
</xsl:template>
```

XSLT – Other Features

- <sort>
  - Supports arranging of elements in different order
    - As child of <xsl:for-each>, <xsl:apply-templates>
- <number>
  - Inserts formatted integer numbers in output document
- Named templates
  - Parameterized processing
  - Like a subroutine call
  - Recursion is possible and important
- <include>, <import>

XSL:FO

- XSL – Formatting Objects
  - XML vocabulary
  - for Formatting documents (layout)
  - Page oriented
  - >50 elements defined for layouting
    - Similar to what word-processors use
- Idea
  - Document content is written in XML
    - Without considering layout
  - Transformed to XSL:FO file with XSLT
    - XSLT adds layout to document

XSL:FO

- XSL:FO Renderer
  - Transforms XSL:FO file into other formats
    - Eg. PDF (Apache FOP)
    - RTF, Latex
  - Used by publishers
- XSL:FO Formatting model
  - Content broken in pages
  - Each contains number of areas
  - Similarities to RTF