Network Services

XML & Web 2.0

Johann Oberleitner

Agenda

- XML
- DTD & XML Schema
- XPath & XSL
- Web 2.0
  - Ajax

HTML

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

- Every technician should be aware of HTML
  - Especially computer scientists

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
  - Continuously 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

**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 / Example 1**

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

**XML / Example 2**

```xml
Whitespaces don't matter:
<Person>
  <Nachname>Mustermann</Nachname>
  <Vorname>Vorname</Vorname>
  <Adresse>
    <Strasse>Argentinierstrasse 8</Strasse>
    <Ort>Wien</Ort>
    <PLZ>1040</PLZ>
  </Adresse>
</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
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE students="students.dtd">
<students>
  <student matnr="e8888888">
    <lastname>Meier</lastname>
    <firstname>Klara</firstname>
    <address/>
  </student>
</students>
```

XML Parts

- XML Preamble
  - Not required, highly recommended
  - "1.0" fixed version
  - 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
  - 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
    - <, >, &, #, &apos,
- Comments
  - <!-- an XML comment -->

DTD / Schema

- Valid XML documents
  - Well-Formed & conforms to rules in DTD or Schema
  - An application may require 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
  - not XML
- 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

DTD Sample

```xml
<!ELEMENT students(student+)>  
<!ELEMENT student(lastname,firstname,adress)>  
<!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>
```
**XML Schema**

- 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!
  - Entities in DTDs are like macros

**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**

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

**XML – Valid & Wellformed**

- **Wellformed**
  - Minimal Requirements for "good" XML document
    - syntactic correctness
    - 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
- **Valid**
  - XML file conforms to one particular DTD or XML Schema file
  - structural correctness
    - No elements that are not defined within Schema
    - Correct order, correct attributes, …
Cascading Style Sheets - CSS
- Allows attachment of style information to HTML
- Modifies Layout of Input elements
- Nesting / Cascading of stylesheets
- External vs. Internal
- May also be applied to XML files!
  - Eg. Automatic rendering of XML files in tabular form (instead of tree)

CSS - Structure
- Generic Structure for Styles
  Selector { Property: Value }
  - Selector specifies class that is modified
  - Property denotes a particular property which value is modified

CSS - Sample
HTML: <body bgcolor="#FF0000">

CSS: body { background-color: #FF0000; }

CSS – Used within HTML
1. In-line
   - Using style attribute in arbitrary HTML tags
     - <body style="background-color: #FF0000;">
  2. Internal
   - Using style tag in HTML header (e.g. after <title></title>) that contains whole CSS
     - <style type="text/css">
       Body { background-color: #FF0000; }
     </style>
  3. External
   - Link to a style sheet in HTML header (after <title></title>)
     - Example
       <head>
         <link rel="stylesheet" type="text/css" href="style/style.css"/>
       </head>

Selector | Property | Value
Cascading Style Sheets - CSS

- Allows attachment of style information to HTML
- Modifies Layout of Input elements
- Nesting / Cascading of stylesheets
- External vs. Internal
- May also be applied to XML files!
  - Eg. Automatic rendering of XML files in tabular form (instead of tree)

XSL

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

XPath

- Selection and addressing language
  - for XML (of course)
  - Based on XML's tree structure
- Result of 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>Torvalds</lastname>
  </student>
</students>
```

/ /student /student/lastname //lastname /students/*/lastname /students/*/lastname/../student[@matnr='7523333']/lastname
**XPath - Axes**

- Navigation within XML tree with so-called 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**

- Predicate within [ ] evaluated relative to a node expression
  - `/student/[predicate]/lastname`
- Multiple predicates in one expression
  - `/student[@matnr='7523333']/name[@nametype='first']`
- Attribute testing by value good
- Element testing by value may be difficult because of whitespaces

**XPath – Selecting other nodes**

- Text Nodes: `text()`
  - Example: `/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 – Expressions and Functions

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

XPath – Datatypes

- Boolean & Numbers
  - Boolean values
    - Predefined: true & false
    - Results of relational operators (==, !=, <, >, <=, =>)
    - Use &lt; instead of <
  - 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
  - Written in XML
- Input is XML
- Output may be
  - XML
  - Text
  - HTML
  - Other formats supported via extensions
- Rule based
  - Rules are matched against input
XSLT Transformation

Input File(s)
XML

XSLT Processor

XSLT Stylesheet

Output File

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

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:apply-templates select="student"/>
  </xsl:template>
</xsl:stylesheet>
```

XSLT - Default Rules

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

XSLT Sample – Generate HTML

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

XSLT Sample – Resulting HTML

- Gates
- Torvalds
XSLT Choices

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

- `<xsl:choose>`
  - Like statements in 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

- `<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 – Other Features / 1

- `<variable>`
  - Supports declaration of variables that refer to xpath expressions
  - Variables can be reused with $varname

- `<for-each>`
  - Supports iteration
  - Over xpath expressions

XSLT – Other Features / 2

- `<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 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

Web 2.0

- Web is currently moving to
  - Rich clients
    - Real applications that run in browsers
  - Support for cooperation of Web applications

Problem of "Web1.0" applications

- Each get/post HTTP request
  - Sends a request to Web Server
  - Waits until response comes back from Web Server
  - Until the response comes back the browser blocks working with Web applications
AJAX - Asynchronous JavaScript

- Solves this problem
- by sending requests in the background
- Waits for answers in the background
- Updates the screen asynchronously
  - End users don’t have to wait until page is reloaded

AJAX - Key Components

- JavaScript
  - Embedded in HTML pages
  - Executed in the Web browser at the client
  - Supports quicker UI interaction mechanisms in the browser
  - without interaction with the Web server
- DOM Tree
  - (X)HTML is modified directly by JavaScript
- CSS
- XMLHttpRequest
  - JavaScript object that supports submitting HTTP Requests asynchronously

Ajax - Principle

- HTTP Server loads main page via HTTP
- Requests other information from Server via HTTP
- JavaScript Startup Code registers JavaScript functions as notification handlers
  - Being called when HTML hyperlinks or HTML form elements are clicked/used
    - Example: Text is entered in a text field
    - Example: Hyperlink is clicked
- Result: JavaScript handler is invoked when hyperlink clicked, form element is used
Ajax – Working Principle / 2
- JavaScript notification handler is invoked synchronously by browser
  - As in rich GUI applications
  - Uses XMLHttpRequest object to setup a HTTP request
    - Often a Web Service is called via SOAP
    - But may just be a request to a Web page
  - Registers another JavaScript function as a HTTP response notification handler
    - A different function is used!
- The HTTP request is sent asynchronously
  - Notification handler for the GUI elements is returned after starting the HTTP request
  - User can continue working in the browser

Ajax – Working Principle / 3
- Some time later the HTTP request is received by the Web Server
  - Sends a response
  - Response comes to the XMLHttpRequest object
    - Processes the response asynchronously
    - Invokes the previously registered Response notification handler

Ajax – Working Principle / 4
- Response notification handler
  - Modifies DOM tree (=XML tree) of the HTML document currently displayed in the browser
  - Supports asynchronous modification of the GUI without stopping the end user in working with the currently displayed window

Web 2.0 / Other developments
- RSS
  - Really Simple Syndication (RSS 2.0)
  - Rich Site Summary (RSS 0.91, RSS 1.0)
  - RDF Site Summary (RSS 0.9, 1.0)
- Goal
  - Sharing news
  - Subscription to parts of web pages
  - So-called Feeds are sent when web page changes
- XML based
  - RSS 1.0 – based on RDF (resource description framework)
  - RSS 2.0 – not based on RDF(!)
- Feed readers may be used to read this news
Summary

- XML
  - Ascii of 21st century
- XPath & XSLT
- Web 2.0 Technologies