合肥學院
HEFEI UNIVERSITY

Distributed Computing
Lesson 18: XML Processing

Thomas Weise · 汤卫思

tweise@hfuu.edu.cn · http://www.it-weise.de

Hefei University, South Campus 2
Faculty of Computer Science and Technology
Institute of Applied Optimization
230601 Shushan District, Hefei, Anhui, China
Econ. & Tech. Devel. Zone, Jinxiu Dadao 99

合肥学院 南艳湖校区/南2区
计算机科学与技术系
应用优化研究所
中国 安徽省 合肥市 蜀山区 230601
经济技术开发区 锦绣大道99号

Outline

Overview

- Now that we can create XML documents and define XML Schemas regarding how they look like...
- ...we want to process these documents from Java.

XML Processing

- Parsing: Read an XML document from a stream

XML Processing

- Parsing: Read an XML document from a stream
- Serializing: Write an XML document to a stream

XML Processing

- Parsing: Read an XML document from a stream
- Serializing: Write an XML document to a stream
- Transforming: Translating an XML document from one form to another one (e.g., with XSLT [1, 2])

XML Parsing

- Non-validating parser: only checks for well-formedness

XML Parsing

- Non-validating parser: only checks for well-formedness
- Validating parser: also checks validity

XML Parsing

- Non-validating parser: only checks for well-formedness
- Validating parser: also checks validity (needs some language definition, e.g., DTD or XSD [3–8])
- Different parsing modes

XML Parsing

- Non-validating parser: only checks for well-formedness
- Validating parser: also checks validity (needs some language definition, e.g., DTD or XSD [3–8])
- Different parsing modes:
  - Result = tree data structure

XML Parsing

- Non-validating parser: only checks for well-formedness
- Validating parser: also checks validity (needs some language definition, e.g., DTD or XSD [3–8])
- Different parsing modes:
  - Result = tree data structure
  - Result = Series of events

XML Parsing

- Non-validating parser: only checks for well-formedness
- Validating parser: also checks validity (needs some language definition, e.g., DTD or XSD [3–8])
- Different parsing modes:
  - Result = tree data structure
  - Result = Series of events:
    - Push-parsing: SAX

XML Parsing

- Non-validating parser: only checks for well-formedness
- Validating parser: also checks validity (needs some language definition, e.g., DTD or XSD [3–8])
- Different parsing modes:
  - Result = tree data structure
  - Result = Series of events:
    - Push-parsing: SAX
    - Pull-parsing: StAX

DOM Trees

- DOM (Document Object Model): object-oriented approach – XML documents represented as tree data structures

DOM Trees

- DOM (Document Object Model): object-oriented approach – XML documents represented as tree data structures
- Hierarchy of document nodes: document, element, attribute, text, note, ...

DOM Trees

- DOM (Document Object Model): object-oriented approach – XML documents represented as tree data structures
- Hierarchy of document nodes: document, element, attribute, text, note, ...
- DOM-API: language independent, standardized at W3C [9]

DOM Trees

- DOM (Document Object Model): object-oriented approach – XML documents represented as tree data structures
- Hierarchy of document nodes: document, element, attribute, text, note, ...
- DOM-API: language independent, standardized at W3C [9]
- Allows for high-level queries and manipulations

DOM Trees

- DOM (Document Object Model): object-oriented approach – XML documents represented as tree data structures
- Hierarchy of document nodes: document, element, attribute, text, note, ...
- DOM-API: language independent, standardized at W3C [9]
- Allows for high-level queries and manipulations, e.g.,
  - traverse nodes in certain order

DOM Trees

- DOM (Document Object Model): object-oriented approach – XML documents represented as tree data structures
- Hierarchy of document nodes: document, element, attribute, text, note, ...
- DOM-API: language independent, standardized at W3C [9]
- Allows for high-level queries and manipulations, e.g.,
  - traverse nodes in certain order
  - add, remove, insert, modify nodes, elements, attributes, ...

DOM Trees

- DOM (Document Object Model): object-oriented approach – XML documents represented as tree data structures
- Hierarchy of document nodes: document, element, attribute, text, note, ...
- DOM-API: language independent, standardized at W3C [9]
- Allows for high-level queries and manipulations, e.g.,
  - traverse nodes in certain order
  - add, remove, insert, modify nodes, elements, attributes, ...
  - finding nodes with specific names and/or features

DOM Trees

- DOM (Document Object Model): object-oriented approach – XML documents represented as tree data structures
- Hierarchy of document nodes: document, element, attribute, text, note, ...
- DOM-API: language independent, standardized at W3C [9]
- Allows for high-level queries and manipulations, e.g.,
  - traverse nodes in certain order
  - add, remove, insert, modify nodes, elements, attributes, ...
  - finding nodes with specific names and/or features
  - ...

DOM Trees

- DOM (Document Object Model): object-oriented approach – XML documents represented as tree data structures
- Hierarchy of document nodes: document, element, attribute, text, note, ...
- DOM-API: language independent, standardized at W3C [9]
- Allows for high-level queries and manipulations, e.g.,
  - traverse nodes in certain order
  - add, remove, insert, modify nodes, elements, attributes, ...
  - finding nodes with specific names and/or features
  - ...
- Complete document must be read before it can be processed

DOM Trees

- DOM (Document Object Model): object-oriented approach – XML documents represented as tree data structures
- Hierarchy of document nodes: document, element, attribute, text, note, ...
- DOM-API: language independent, standardized at W3C [9]
- Allows for high-level queries and manipulations, e.g.,
  - traverse nodes in certain order
  - add, remove, insert, modify nodes, elements, attributes, ...
  - finding nodes with specific names and/or features
  - ...
- Complete document must be read before it can be processed
- Everything must be in memory (high memory consumption)

DOM Document Creation

Listing: Creating and writing a DOM document (DOMCreateAndPrintDocument.java).

```
import org.w3c.dom.Document; import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Element;  import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Node;     import javax.xml.transform.TransformerFactory;
import org.w3c.dom.NodeList; import javax.xml.parsers.DocumentBuilderFactory;
import java.io.File;         import javax.xml.transform.dom.DOMSource;

public class DOMCreateAndPrintDocument {
  public static void main(final String argv[]) throws Throwable {
    Document  document;         Node  n, m;

    document = DocumentBuilderFactory.newInstance()
                                     .newDocumentBuilder()
                                     .newDocument();  // create an empty document

    n = document.createElement("myNewElement");       // create and add a new element
    document.appendChild(n);

    m = document.createElement("myNewSubElement");    // create and add a sub-element
    n.appendChild(m);

    n = document.createAttribute("myAttribute");      // create a new attribute
    n.setNodeValue("myAttributeValue");               // add the attribute
    m.getAttributes().setNamedItem(n);

    TransformerFactory.newInstance()
                      .newTransformer()
                      .transform(new DOMSource(document),
                                 new StreamResult(System.out)); // print to stdout
  }
}
```

Reading and Modifying DOM Documents

Listing: Reading, Modifying, and writing a DOM document (DOMReadAndPrintDocument.java).

```
import org.w3c.dom.Document; import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Element;  import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.Node;     import javax.xml.transform.TransformerFactory;
import org.w3c.dom.NodeList; import javax.xml.parsers.DocumentBuilderFactory;
import java.io.File;         import javax.xml.transform.dom.DOMSource;

public class DOMReadAndPrintDocument {
  public static void main(final String argv[]) throws Throwable {
    Document  document;         Node  n, m;

    document = DocumentBuilderFactory.newInstance()       // read the course example
                                     .newDocumentBuilder()// document as DOM tree
                                     .parse("../xml/course.xml");

    n = document.createElement("myNewElement");       // create and add a new element
    document.getDocumentElement().appendChild(n);

    m = document.createElement("myNewSubElement");    // create and add a sub-element
    n.appendChild(m);

    n = document.createAttribute("myAttribute");      // create a new attribute
    n.setNodeValue("myAttributeValue");               // add the attribute
    m.getAttributes().setNamedItem(n);


    TransformerFactory.newInstance()
                      .newTransformer()
                      .transform(new DOMSource(document),
                                 new StreamResult(System.out)); // print to stdout
  }
}
```

SAX Parsing

- SAX = Simple API for XML [10]

SAX Parsing

- SAX = Simple API for XML [10]
- SAX parser receives implementation of event handler interface

SAX Parsing

- SAX = Simple API for XML [10]
- SAX parser receives implementation of event handler interface
- SAX parser processes a stream from begining to end

SAX Parsing

- SAX = Simple API for XML [10]
- SAX parser receives implementation of event handler interface
- SAX parser processes a stream from begining to end
- Notifies event handler of events such as begin and end of elements

SAX Parsing

- SAX = Simple API for XML [10]
- SAX parser receives implementation of event handler interface
- SAX parser processes a stream from begining to end
- Notifies event handler of events such as begin and end of elements
- Advantages

SAX Parsing

- SAX = Simple API for XML [10]
- SAX parser receives implementation of event handler interface
- SAX parser processes a stream from begining to end
- Notifies event handler of events such as begin and end of elements
- Advantages:
  - stream processing is easy and fast

SAX Parsing

- SAX = Simple API for XML [10]
- SAX parser receives implementation of event handler interface
- SAX parser processes a stream from begining to end
- Notifies event handler of events such as begin and end of elements
- Advantages:
  - stream processing is easy and fast
  - no tree needs to be built in memory

SAX Parsing

- SAX = Simple API for XML [10]
- SAX parser receives implementation of event handler interface
- SAX parser processes a stream from begining to end
- Notifies event handler of events such as begin and end of elements
- Advantages:
  - stream processing is easy and fast
  - no tree needs to be built in memory
  - documents can be parsed partially

SAX Parsing

- SAX = Simple API for XML [10]
- SAX parser receives implementation of event handler interface
- SAX parser processes a stream from begining to end
- Notifies event handler of events such as begin and end of elements
- Advantages:
  - stream processing is easy and fast
  - no tree needs to be built in memory
  - documents can be parsed partially
  - uninteresting stuff can be ignored

SAX Parsing

- SAX = Simple API for XML [10]
- SAX parser receives implementation of event handler interface
- SAX parser processes a stream from begining to end
- Notifies event handler of events such as begin and end of elements
- Advantages:
  - stream processing is easy and fast
  - no tree needs to be built in memory
  - documents can be parsed partially
  - uninteresting stuff can be ignored
- Disadvantages

SAX Parsing

- SAX = Simple API for XML [10]
- SAX parser receives implementation of event handler interface
- SAX parser processes a stream from begining to end
- Notifies event handler of events such as begin and end of elements
- Advantages:
  - stream processing is easy and fast
  - no tree needs to be built in memory
  - documents can be parsed partially
  - uninteresting stuff can be ignored
- Disadvantages:
  - no complex queries or processing of elements possible

SAX Parsing

- SAX = Simple API for XML [10]
- SAX parser receives implementation of event handler interface
- SAX parser processes a stream from begining to end
- Notifies event handler of events such as begin and end of elements
- Advantages:
  - stream processing is easy and fast
  - no tree needs to be built in memory
  - documents can be parsed partially
  - uninteresting stuff can be ignored
- Disadvantages:
  - no complex queries or processing of elements possible
  - applications possible need to build their own parse tree

Parsing with SAX

Listing: Parsing with SAX (SAXReaderExample.java).

```
import javax.xml.parsers.SAXParser;   import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;        import org.xml.sax.helpers.DefaultHandler;
import org.w3c.dom.Document;

public class SAXReaderExample extends DefaultHandler { //derive new class from
    DefaultHandler

  public static void main(final String argv[]) throws Throwable {
    SAXParserFactory.newInstance()
                    .newSAXParser()
                    .parse("../xml/courseWithNamespace.xml", // parse the course document
                           new SAXReaderExample());
  }

  // implement one of the many event handler methods
  public void startElement(String uri, String localName, String qName, Attributes
      attributes) {
    System.out.print("<" + qName);
    for(int i = attributes.getLength(); (--i) >= 0; ) {
      System.out.print("␣" + attributes.getQName(i) + "=" + attributes.getValue(i));
    }
    System.out.println(">");
  }
}
```

Parsing and Validating with SAX

Listing: Parsing with SAX with validation (SAXReaderExampleValidating.java).

```
import javax.xml.parsers.SAXParser;   import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;        import org.xml.sax.helpers.DefaultHandler;
import org.w3c.dom.Document;

public class SAXReaderExampleValidating extends DefaultHandler { //derive new class from
    DefaultHandler

  public static void main(final String argv[]) throws Throwable {
    SAXParserFactory factory = SAXParserFactory.newInstance();
    factory.setValidating(true); // validate the document
    factory.newSAXParser()
           .parse("../xml/courseWithNamespaceAndSchemaLocation.xml", // parse the document
                  new SAXReaderExampleValidating());
  }

  // implement one of the many event handler methods
  public void startElement(String uri, String localName, String qName, Attributes
      attributes) {
    System.out.print("<" + qName);
    for(int i = attributes.getLength(); (--i) >= 0; ) {
      System.out.print("␣" + attributes.getQName(i) + "=" + attributes.getValue(i));
    }
    System.out.println(">");
  }
}
```

StAX Parsing

- StAX = Streaming API for XML [11]

StAX Parsing

- StAX = Streaming API for XML [11]
- Stream pull parser, similar to SAX

StAX Parsing

- StAX = Streaming API for XML [11]
- Stream pull parser, similar to SAX, but
- Programmer actually pulls events from parser, whereas SAX parsers actively invoke event handlers

StAX Parsing

- StAX = Streaming API for XML [11]
- Stream pull parser, similar to SAX, but
- Programmer actually pulls events from parser, whereas SAX parsers actively invoke event handlers
- Same advantages and disadvantages as SAX, but more decision power for programmer

Parsing with StAX

Listing: Parsing with StAX (StAXReaderExample.java).

```
import java.io.File;                          import java.io.FileReader;
import javax.xml.stream.XMLInputFactory;      import javax.xml.stream.XMLStreamReader;
import javax.xml.stream.XMLStreamConstants;

public class StAXReaderExample {
  @SuppressWarnings("incomplete-switch")
  public static void main(final String argv[]) throws Throwable {
    XMLInputFactory factory;     XMLStreamReader reader;

    factory = XMLInputFactory.newInstance(); //create the necessary objects
    try(final FileReader fr = new FileReader("../xml/courseWithNamespace.xml")) {
      reader  = factory.createXMLStreamReader(fr); //create the parser...

      while (reader.hasNext()) { // iterate over the "XML-events"...

        switch (reader.next()) { // get the next event and switch/case the event type
          case XMLStreamConstants.START_ELEMENT: {          // element start?
            System.out.println("<" + reader.getLocalName() + ">"); break;
          }
          case XMLStreamConstants.END_ELEMENT: {            // element end?
            System.out.println("</" + reader.getLocalName() + ">");
          }
        }
      }
    }
  }
}
```

Listing: Writing with StAX (StAXWriterExample.java).

```
import javax.xml.stream.XMLOutputFactory;     import javax.xml.stream.XMLStreamWriter;

public class StAXWriterExample {
  public static void main(final String argv[]) throws Throwable {
    XMLOutputFactory output;     XMLStreamWriter writer;

    String namespace = "ustc:courses";                  // let's use the course namespace
    output = XMLOutputFactory.newInstance();            // instantiate the necessary
        classes
    writer = output.createXMLStreamWriter(System.out); // create a stream writer (to
        stdout)

    writer.writeStartDocument();                        // begin an xml document
    writer.setPrefix("c", namespace);                   // define a namespace prefix

    writer.writeStartElement(namespace, "course");      // write the root element
      writer.writeNamespace("c", namespace);            // write the namespace prefix
      writer.writeAttribute(namespace,  "courseName", "Distributed Computing");

      writer.writeStartElement(namespace, "units");     // start an element
        writer.writeCharacters("60");                   // write some text
      writer.writeEndElement();                         // close/end an element

      writer.writeStartElement(namespace, "teachers"); // start a teachers element
        writer.writeStartElement(namespace, "teacher");// start a teacher element
          writer.writeAttribute(namespace, "familyName",   "Weise"); //write some
              attribute
        writer.writeEndElement();                       // end the teacher element
      writer.writeEndElement();                         // end the teachers element

    writer.writeEndElement();                           // end the course element
    writer.flush();                                     // flush the data
  }
}
```

Advantages of StAX/SAX Parsing

- Smaller memory footprint

Advantages of StAX/SAX Parsing

- Smaller memory footprint
- We do not need to load the complete document, but can stop after we got all the data we are interested in

Advantages of StAX/SAX Parsing

- Smaller memory footprint
- We do not need to load the complete document, but can stop after we got all the data we are interested in
- With StAX, we can already write some part of the output even *before* having read any of the input

Advantages of StAX/SAX Parsing

- Smaller memory footprint
- We do not need to load the complete document, but can stop after we got all the data we are interested in
- With StAX, we can already write some part of the output even *before* having read any of the input
- We can keep writing more output while reading and processing input

Advantages of StAX/SAX Parsing

- Smaller memory footprint
- We do not need to load the complete document, but can stop after we got all the data we are interested in
- With StAX, we can already write some part of the output even *before* having read any of the input
- We can keep writing more output while reading and processing input
- This is especially interesting for distributed systems: We begin sending the response while receiving request

Advantages of StAX/SAX Parsing

- Smaller memory footprint
- We do not need to load the complete document, but can stop after we got all the data we are interested in
- With StAX, we can already write some part of the output even *before* having read any of the input
- We can keep writing more output while reading and processing input
- This is especially interesting for distributed systems: We begin sending the response while receiving request
- Axis (see next lesson) uses StAX under the hood

Advantages of StAX/SAX Parsing

- Smaller memory footprint
- We do not need to load the complete document, but can stop after we got all the data we are interested in
- With StAX, we can already write some part of the output even *before* having read any of the input
- We can keep writing more output while reading and processing input
- This is especially interesting for distributed systems: We begin sending the response while receiving request
- Axis (see next lesson) uses StAX under the hood
- The DOM API can be implemented by using SAX/StAX to build the DOM tree...

Stylesheet Transformation (XSL)

- In many situations, we want to transform data in one XML dialect (schema) to another format (or other XML dialect)

Stylesheet Transformation (XSL)

- In many situations, we want to transform data in one XML dialect (schema) to another format (or other XML dialect)
- Example: XML defines no presentation

Stylesheet Transformation (XSL)

- In many situations, we want to transform data in one XML dialect (schema) to another format (or other XML dialect)
- Example: XML defines no presentation:
  - idea: transform XML to HTML [12] or XHTML [13] for presentation in browser

Stylesheet Transformation (XSL)

- In many situations, we want to transform data in one XML dialect (schema) to another format (or other XML dialect)
- Example: XML defines no presentation:
  - idea: transform XML to HTML [12] or XHTML [13] for presentation in browser
  - or to text

Stylesheet Transformation (XSL)

- In many situations, we want to transform data in one XML dialect (schema) to another format (or other XML dialect)
- Example: XML defines no presentation:
  - idea: transform XML to HTML [12] or XHTML [13] for presentation in browser
  - or to text
  - or to LaTeX [14, 15] ...

Stylesheet Transformation (XSL)

- In many situations, we want to transform data in one XML dialect (schema) to another format (or other XML dialect)
- Example: XML defines no presentation:
  - idea: transform XML to HTML [12] or XHTML [13] for presentation in browser
  - or to text
  - or to LaTeX [14, 15] ...
- Example: Translate message from one XML-based protocol to another

Stylesheet Transformation (XSL)

- In many situations, we want to transform data in one XML dialect (schema) to another format (or other XML dialect)
- Example: XML defines no presentation:
  - idea: transform XML to HTML [12] or XHTML [13] for presentation in browser
  - or to text
  - or to LaTeX [14, 15] ...
- Example: Translate message from one XML-based protocol to another
- Extensible Stylesheet Language (XSL [2]) Transformation (XSLT [1])

Stylesheet Transformation (XSL)

- In many situations, we want to transform data in one XML dialect (schema) to another format (or other XML dialect)
- Example: XML defines no presentation:
  - idea: transform XML to HTML [12] or XHTML [13] for presentation in browser
  - or to text
  - or to LaTeX [14, 15] ...
- Example: Translate message from one XML-based protocol to another
- Extensible Stylesheet Language (XSL [2]) Transformation (XSLT [1])
  - transformation language with elements from functional programming languages

Stylesheet Transformation (XSL)

- In many situations, we want to transform data in one XML dialect (schema) to another format (or other XML dialect)
- Example: XML defines no presentation:
  - idea: transform XML to HTML [12] or XHTML [13] for presentation in browser
  - or to text
  - or to LaTeX [14, 15] ...
- Example: Translate message from one XML-based protocol to another
- Extensible Stylesheet Language (XSL [2]) Transformation (XSLT [1])
  - transformation language with elements from functional programming languages
  - template-based and declarative

Stylesheet Transformation (XSL)

- In many situations, we want to transform data in one XML dialect (schema) to another format (or other XML dialect)
- Example: XML defines no presentation:
  - idea: transform XML to HTML [12] or XHTML [13] for presentation in browser
  - or to text
  - or to LaTeX [14, 15] ...
- Example: Translate message from one XML-based protocol to another
- Extensible Stylesheet Language (XSL [2]) Transformation (XSLT [1])
  - transformation language with elements from functional programming languages
  - template-based and declarative
  - uses XPath [16] for navigation in document tree

XSLT Style Sheet

Listing: An XSLT style sheet translating courses.xsd-compliant files to HTML (courses2html.xslt).

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                              xmlns:c  ="ustc:courses"
                              exclude-result-prefixes="c␣xsl">
  <xsl:output method="html" indent="yes"/>
  <xsl:strip-space elements="*"/>

  <xsl:template match="c:course">
    <html>
      <body>
        <h1><xsl:value-of select="@courseName" /> (<xsl:value-of select="c:units" />)</h1>
        <xsl:apply-templates select="c:teachers"/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="c:teachers">
    <ul>
      <xsl:for-each select="c:teacher">
        <li><xsl:value-of select="@familyName" />, <xsl:value-of select="@personalName"
            /></li>
      </xsl:for-each>
    </ul>
  </xsl:template>
</xsl:stylesheet>
```

XSLT Transformation

Listing: Java program performing the transformation (XSLTTransform.java).

```
import javax.xml.transform.stream.StreamSource; import javax.xml.transform.Result;
import javax.xml.transform.stream.StreamResult; import javax.xml.transform.Source;
import javax.xml.transform.TransformerFactory;  import java.io.File;
import javax.xml.transform.Transformer;

public class XSLTTransform {
  public static void main(final String argv[]) throws Throwable {
    TransformerFactory factory;    Transformer transformer;
    Source xmlInput, xsltInput;    Result      output;


    xmlInput    = new StreamSource(new File("../xml/courseWithNamespace.xml"));
    xsltInput   = new StreamSource(new File("../xml/courses2html.xslt"));
    output      = new StreamResult(System.out);

    factory     = TransformerFactory.newInstance();  // create the necessary objects to
    transformer = factory.newTransformer(xsltInput); // transform an XML stream with XSLT!

    transformer.transform(xmlInput, output);         // flush the data
  }
}
```

The Resulting HTML

Listing: The resulting HTML file (course.html).

```
<html xmlns:c="http://www.test.com/courses.xsd">
<body>
<h1>Distributed Computing (60)</h1>
<ul>
<li>Weise, Thomas</li>
<li>Chen, Xianglan</li>
</ul>
</body>
</html>
```

Summary

- XML is predominant data format in internet, basis for XHTML [13] and many protocols
- XML dialects can be specified with XML Schemas [3–5, 17]
- Broad support, many processing and transformation tools (DOM [9], SAX [10], StAX [11])
- Transformations via XSLT [1, 2]
- Core of what we will learn in the next lesson: Web Services!

谢谢

Thank you

Thomas Weise [汤卫思]
tweise@hfuu.edu.cn
http://www.it-weise.de

Hefei University, South Campus 2
Institute of Applied Optimization
Shushan District, Hefei, Anhui,
China

Caspar David Friedrich, “Der Wanderer über dem Nebelmeer”, 1818
http://en.wikipedia.org/wiki/Wanderer_above_the_Sea_of_Fog

Bibliography

Bibliography I

1. James Clark. *XSL Transformations (XSLT)*. W3C Recommendation. MIT/CSAIL (USA), ERCIM (France), Keio University (Japan): World Wide Web Consortium (W3C), version 1 edition, November 16, 1999. URL http://www.w3.org/TR/1999/REC-xslt-19991116.
2. Anders Berglund, editor. *Extensible Stylesheet Language (XSL) Version 1.1*. W3C Recommendation. MIT/CSAIL (USA), ERCIM (France), Keio University (Japan): World Wide Web Consortium (W3C), December 5, 2006. URL http://www.w3.org/TR/2006/REC-xsl11-20061205/.
3. Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn, editors. *XML Schema Part 1: Structures Second Edition*. W3C Recommendation. MIT/CSAIL (USA), ERCIM (France), Keio University (Japan): World Wide Web Consortium (W3C), October 28, 2004. URL http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/.
4. Paul V. Biron and Ashok Malhotra, editors. *XML Schema Part 2: Datatypes Second Edition*. W3C Recommendation. MIT/CSAIL (USA), ERCIM (France), Keio University (Japan): World Wide Web Consortium (W3C), October 28, 2004.
5. Cliff Binstock. *The XML Schema – Complete Reference*. Reading, MA, USA: Addison-Wesley Professional, 2003. ISBN 0672323745 and 9780672323744. URL http://books.google.de/books?id=pCLJly0eZwEC.
6. Marco Skulschus and Marcus Wiederstein. *XML Schema*. Berlin, Germany: Comelio GmbH, 2008. ISBN 3898424723, 393970122X, 9783898424721, and 9783939701224. URL http://books.google.de/books?id=Vi6dTuL6g7kC.
7. Eric Van der Vlist. *XML Schema – The W3C's Object-Oriented Descriptions for XML*. O'Reilly Series. Sebastopol, CA, USA: O'Reilly Media, Inc., 2002. ISBN 0596002521 and 9780596002527. URL http://books.google.de/books?id=2h8T6xqKeVUC.
8. Jack Lindsey. Subtyping in w3c xml schema, part 1 – three degrees of inheritance. *The Data Administration Newsletter – TDAN.com*, 12(4), April 3, 2008. URL http://www.tdan.com/view-articles/7185.
9. Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Arnaud Le Hors, Gavin Nicol, Jonathan Robie, Peter Sharpe, Bill Smith, Jared Sorensen, Robert Sutor, Ray Whitmer, and Chris Wilson. *Document Object Model (DOM) Level 1 Specification*. W3C Recommendation. MIT/CSAIL (USA), ERCIM (France), Keio University (Japan): World Wide Web Consortium (W3C), 1 edition, October 1, 1998. URL http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001.
10. David Brownell. *SAX2 – Processing XML Efficiently with Java*. Sebastopol, CA, USA: O'Reilly Media, Inc., 2002. ISBN 0596002378 and 9780596002374. URL http://books.google.de/books?id=Gp4chqpT4vkC.
11. *JSR 173: Streaming API for XML*. Java Specification Requests (JSR). San José, CA, USA: BEA Systems, Inc., final v1.0 edition, October 8, 2003. URL http://download.oracle.com/otndocs/jcp/streaming_xml-1.0-fr-spec-oth-JSpec/.

Bibliography II

12. Dave Raggett, Arnaud Le Hors, and Ceriel J. H. Jacobs. *HTML 4.01 Specification*. W3C Recommendation. MIT/CSAIL (USA), ERCIM (France), Keio University (Japan): World Wide Web Consortium (W3C), December 24, 1999. URL http://www.w3.org/TR/1999/REC-html401-19991224.
13. Murray Altheim and Shane McCarron. *XHTML™ 1.1 – Module-based XHTML – Second Edition*. W3C Recommendation. MIT/CSAIL (USA), ERCIM (France), Keio University (Japan): World Wide Web Consortium (W3C), November 23, 2010. URL http://www.w3.org/TR/2010/REC-xhtml11-20101123.
14. Leslie Lamport. *LaTeX: A Document Preparation System. User's Guide and Reference Manual*. Reading, MA, USA: Addison-Wesley Publishing Co. Inc., 1994. ISBN 0201529831 and 9780201529838. URL http://books.google.de/books?id=19pzDwEACAAJ.
15. Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Tools and Techniques for Computer Typesetting. Reading, MA, USA: Addison-Wesley Publishing Co. Inc., 1994. ISBN 0201541998 and 9780201541991. URL http://books.google.de/books?id=54A3MuBzIrEC.
16. Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon, editors. *XML Path Language (XPath) 2.0 (Second Edition)*. W3C Recommendation. MIT/CSAIL (USA), ERCIM (France), Keio University (Japan): World Wide Web Consortium (W3C), December 14, 2010. URL ttp://www.w3.org/TR/2010/REC-xpath20-20101214/.
17. David C. Fallside and Priscilla Walmsley, editors. *XML Schema Part 0: Primer Second Edition*. W3C Recommendation. MIT/CSAIL (USA), ERCIM (France), Keio University (Japan): World Wide Web Consortium (W3C), second edition, October 28, 2004. URL http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/.