XML Vorlesung FH Aargau, SS 2006

XML Informationsmodelle

Erik Wilde

3.4.2006

http://dret.net/lectures/xml-fhnw-ss06/

Übersicht

- XML Namespaces für Schema-Kombination
- XML Information Set als abstrakte Sicht
  - Information Items und ihre Properties
  - Canonical XML als Anwendung des XML Infoset
- Programmieren mit XML APIs
  - Document Object Model (DOM)
  - Simple API for XML (SAX)

XML Namespaces

- XML ermöglicht anwendungsabhängige Namen
  - Namenskonflikte sind nahezu vorprogrammiert
  - Problem der "friedlichen Koexistenz" von Namensräumen
- Lösung auf einfache Art: XML Namespaces
  - Identifikation eines Namensraumes durch eine URI
  - Verbindung eines lokalen Präfix mit der globalen URI
  - Verwendung des Präfix im XML-Dokument
- jeder kann eigene Namespaces definieren
  - keine formale Bedeutung der Namespace URI
  - meistens Verweis auf DTD oder Standard-Dokument
- Namespace Support wichtig für XML Parser

Beispiel XML Namespaces

```
<?xml version="1.0"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">

  <xsl:output method="html" indent="yes" />

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

  <xsl:template match="kurs">
    <html>
      <head>
        <title>
        <xsl:value-of select="titel"/>
        </title>
      </head>
```

Anwendungen für Namespaces

- Kombination verschiedener Namensräume
  - potentielle Namenskonflikte (XML hat lokale Namen)
- Schema, das andere Schemas wiederverwendet
  - z.B. "Kurs-DTD" zusammen mit XHTML oder MathML
- Trend zur Modularisierung (XHTML 1.1)
- XSLT zum Namespace-Filtern sehr einfach
  - Node-Tests können Namespaces erkennen
- alle XML-Standards unterstützen Namespaces

Namespaces und XML Processing

- keine eindeutige Zuordnung möglich
  - Namespace durch 0-m Schemas beschrieben
  - Schema kann Namen für 0-n Namespaces definieren
- Normalfall: Schema definiert Namespace
  - `targetNamespace` Attribut des Schemas
- XML Schema benutzt Namespaces
  - für die Elemente und Attribute der Schemasprache
  - für Attribute in Instanzen
  - für *Import* und *Redefine* von Schemas

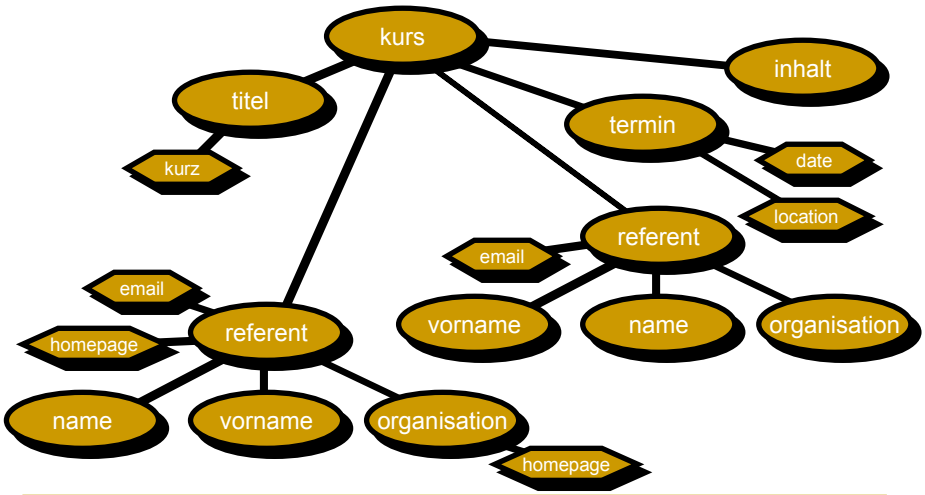
XML Information Set (XML Infoset)

- XML definiert Syntax für strukturierte Daten
  - eigentlich wichtig ist die Struktur
  - die Syntax ist Nebensache und irritiert manchmal
- XML Infoset definierte eine abstrakte Sicht
  - Struktur aus Information Items
  - keine Schnittstelle, sondern nur ein Modell
- OSI taucht immer wieder auf...
  - XML Infoset als ASN.1 abstrakte Syntax
  - XML als (eine mögliche...) Codierung des Baumes

Beispiel (XML)

```
<?xml version="1.0" ?>
<!DOCTYPE kurs SYSTEM "kurs.dtd">

<kurs>
<titel kurz="XML">XML - Grundlagen und Umfeld</titel>

<referent email="xml@dret.net"
          homepage="http://dret.net/">
  <vorname>Erik</vorname>
  <name>Wilde</name>
  <organisation homepage="http://www.tik.ee.ethz.ch/">ETH
  Zürich</organisation>
</referent>

<referent> ... </referent>

<termin date="20000512" location="technopark"/>

<inhalt> ... </inhalt> </kurs>
```

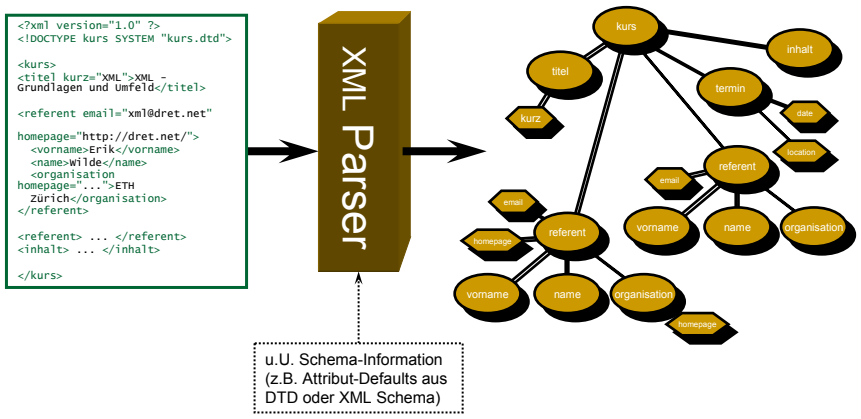
Infoset eines XML Dokuments



Entstehung des Infosets

- mehr als ein Parse Tree
  - XML definiert die XML Syntax in EBNF-Produktionen
  - XML Infoset ist eine Abstraktion der Syntax
    - viele Syntax-Details werden im Infoset ignoriert
    - Teile der Struktur werden interpretiert (z.B. Referenzen)
- ändert sich je nach Parser-Optionen
  - z.B. Attribut-Defaults erscheinen erst in valid Infosets
  - die Typen von Attributen kommen auch aus der DTD
  - es kann verschiedene Schemas geben
    - z.B. eine DTD und ein XML Schema für ein Dokument

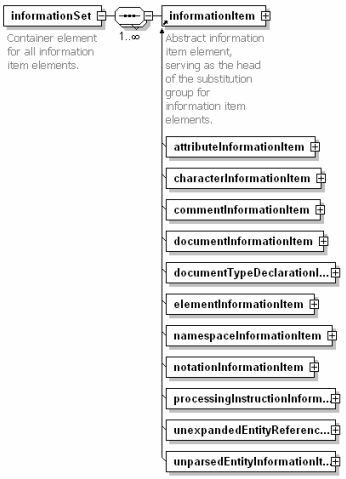
XML Dokumente und Infoset/DOM



XML Infoset Information Items

- 11 Typen von Information Items:
  1. Document Information Item
  2. Element Information Items
  3. Attribute Information Items
  4. Processing Instruction Information Items
  5. Unexpanded Entity Reference Information Items
  6. Character Information Items
  7. Comment Information Items
  8. Document Type Declaration Information Item
  9. Unparsed Entity Information Items
  10. Notation Information Items
  11. Namespace Information Items

Infoset als Menge von Infoset Items



- Infoset als Item Menge
  - keine Reihenfolge
  - keine spezielle Syntax
  - Referenzen aufeinander
- Infoset Standard
  - definiert die Items
  - definiert ihre Properties
  - keine Repräsentation

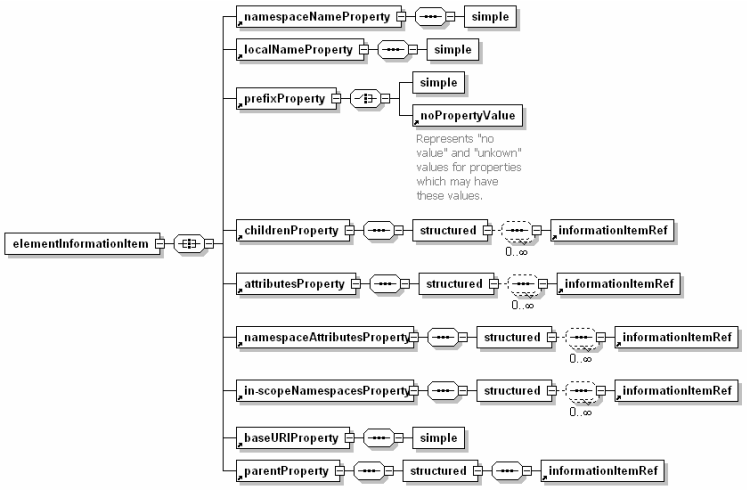
Document Item

- repräsentiert das gesamte Dokument
  - Startpunkt der Hierarchie von Information Items
  - genau ein Document Item pro Dokument
- definiert durch neun Properties
  - untergeordnete Items
    - Children (z.B. Kommentare) und das Document Element
  - Notations und Unparsed Entities der DTD
  - Base URI des Dokuments
  - Information aus der XML Deklaration
    - Character Encoding, Version, Standalone Deklaration
  - Flag das die komplette DTD-Verarbeitung signalisiert

Element Items

- repräsentiert ein Element
  - `<p id="p5">hallo <em>da!</em></p>`
- definiert durch neun Properties
  - Namespace Name, Local Name, Namespace Prefix
  - zugeordnete Items
    - Children (Elemente, Zeichen, Kommentare, ...)
    - Attribute und Attribute mit Namespace Deklarationen
    - gültige Namespaces
  - Base URI des Elements
  - Referenz auf das Eltern-Item (Element/Document)

Element Item (Schema)



Attribute Items

- Attribute sind eigenständige Items
  - sind jeweils genau einem Element zugeordnet
- definiert durch acht Properties
  - Namespace Name, Local Name, Namespace Prefix
  - normalisierter Wert des Attributs + specified Flag
    - Behandlung von Whitespace, Line Endings, Entities
  - Attributtyp (kommt aus dem Schema)
  - Referenzen auf andere Items (z.B. `IDREF` Attribute)
  - Referenz auf das Element des Attributs
- Namespace Deklarationen sind Attribute
  - werden zusätzlich als Namespace Items repräsentiert

Processing Instruction Items

- Platzhalter für eine Processing Instruction
  - `<?xml-stylsheet href="style.css"?>`
  - Inhalt ist beliebig (oft so genannte Pseudo-Attribute)
- definiert durch fünf Properties
  - das Target der PI (der Name nach "`<?`")
  - der Inhalt als String
    - d.h. Pseudo-Attribute müssen "von Hand" interpretiert werden
  - die Base URI der PI
  - Verweis auf die Notation (falls deklariert)
  - Verweis auf das enthaltende Item
    - Document, Element oder Document Type Definition Item

Unexpanded Entity Reference Items

- Platzhalter für ein external parsed Entity
  - `<!ENTITY text SYSTEM "text.xml">`
    `&text;`
  - external Entity muss well-formed XML enthalten
  - validating Processors müssen alle Entities auflösen
- definiert durch fünf Properties
  - der Name des Entities
  - System und Public Identifier (falls vorhanden)
    - die Base URI des System Identifier
  - Verweis auf das enthaltende Item
    - immer ein Element Information Item

Character Items

- für jedes Zeichen in XML Character Data
  - `müde, m&#252;de oder m&#xFC;de`
  - Character Entities repräsentieren ein Zeichen
- jedes Zeichen hat drei Properties
  - der ISO 10646 (Unicode) Character Code
    - erlaubt tausende von Zeichen
  - ein Flag für "Element Content Whitespace"
    - `<name> <vorname>Erik</vorname> </name>`
    - kann nur mit der Elementdeklaration beantwortet werden
  - Verweis auf das enthaltende Item

Comment Items

- repräsentiert Kommentare im Dokument
  - `<!-- Kommentartext -->`
- wird durch zwei Properties dargestellt
  - Text des Kommentares
    - Markup in Kommentaren wird ignoriert
  - Verweis auf das enthaltende Item
    - Document oder Element Information Item
- Kommentare in der DTD werden ignoriert

Document Type Declaration Item

- repräsentiert die DTD eines Dokuments
  - muss nicht existieren (XML ohne DTD ist erlaubt)
  - Teile sind in anderen Items (Entities und Notations)
- definiert durch vier Properties
  - System und Public Identifier (falls vorhanden)
  - Verweise auf alle Processing Instructions in der DTD
    - Verweise auf die jeweiligen Processing Instruction Items
  - ein Verweis auf das Document Item
- damit kein Zugriff auf viele Informationen
  - insbesondere Element- und Attributdeklarationen

Unparsed Entity Items

- für jedes deklarierte unparsed Entity
  - `<!ENTITY pic SYSTEM "pic.gif" NDATA gif87a>`
  - d.h. nicht für Entity-Referenzen
- definiert durch sechs Properties
  - Name des Entities
  - System und Public Identifier (falls vorhanden)
    - die Base URI des System Identifier
  - die Notation des Entities
    - als Name aus der Entity Deklaration
    - als Referenz auf ein Notation Item (falls vorhanden)

Notation Item

- repräsentiert deklarierte Notations
  - Notations deklarieren Formate externer Resourcen
  - `<!NOTATION gif87a SYSTEM "GIF87a.not">`
- definiert durch vier Properties
  - der Name der Notation (z.B. GIF89a)
  - der System Identifier der Notation
    - falls er angegeben wurde
  - der Public Identifier der Notation
    - falls er angegeben wurde
  - die Base URI des System Identifier
    - nur wichtig falls System Identifier vorhanden und relative URI

Namespace Items

- repräsentieren in-Scope Namespaces
  - d.h. alle für ein Element gültigen Namespaces
  - sämtliche Deklarationen auf und "über" dem Element
- definiert durch zwei Properties
  - der Namespace Name (d.h. die URI)
  - der Namespace Prefix
- Namespace-Deklarationen tauchen zweimal auf
  - als Attribute Information Item
  - als Namespace Information Item(s)
    - oftmals mehrfach: auf allen Nachkommen des Elementes

Nicht alles ist im Infoset

- die Ordnung von Attributen im Start-Tag
- Whitespace aus Start- und End-Tags
- Unterscheidung von leeren Elementen
  - `<hallo></hallo>` oder `<hallo/>`
- praktisch alles aus der DTD
  - die Identifier der DTD sind im Infoset
  - Processing Instructions in der DTD sind im Infoset
- Unterschied Zeichen/Character Entity
  - `&#252;` und `&#xFC;` und ü sind nicht unterscheidbar

Nicht alles muss im Infoset sein

- Applikationen sind frei in der Auswahl
  - können Items und/oder Properties weglassen
  - können Items und/oder Properties dazunehmen
- XML Infoset ist ein Punkt in einem Kontinuum
  - ein extrem ist ein zeichengetreues Modell
    - wird z.B. benötigt für XML-Editoren
    - andere Applikationen, die Round-Trips benötigen
  - anderes Extrem ist eine starke Abstraktion
    - wenn man von XML eigentlich gar nichts wissen will
- XML Infoset als vom W3C gesetzte Marke

Canonical XML

- XML ist sehr syntax-zentriert
  - Vergleich von XML-Dokumenten ist schwierig
  - Whitespace, Attribute order, Entities, Namespaces, ...
- Canonical XML legt eine Normalisierung fest
  - mache ein XML Infoset aus dem Dokument
  - generiere aus diesem Baum XML nach festen Regeln
- bessere Basis für Vergleiche als reines XML
  - allerdings aufwendige Operation
  - noch nicht unterstützt von der meisten Software
- interessant auch für digitale Signaturen von XML

APIs: Zugriff auf XML Dokumente

- XML-Dokumente sind strukturierten Daten
  - Problem der definierten Interfaces
  - Zugriff oder Speicherung nicht unbedingt als XML
- XML Information Set: abstrakte Sicht
  - Standards beziehen sich meist auf XML Infoset
  - syntax-unabhängige abstrakte Sicht von XML
  - jedoch: nur ein Modell, keine definierte Schnittstelle!
- Programmierschnittstellen für XML-Dokumente
  - Document Object Model (DOM)
  - Simple API for XML (SAX)

Document Object Model (DOM)

- entstanden für portables JavaScript
- Programmierinterface für Dokumente
- unabhängig vom Dokumentenmodell
  - core definiert einen strukturellen Kern
  - HTML definiert HTML-spezifische Aspekte
  - XML erlaubt den Zugriff auf XML-Dokumente
- unabhängig von der Programmiersprache
  - Definition in Interface Definition Language (IDL)
  - language bindings für diverse Sprachen
  - bisher ECMAScript und Java

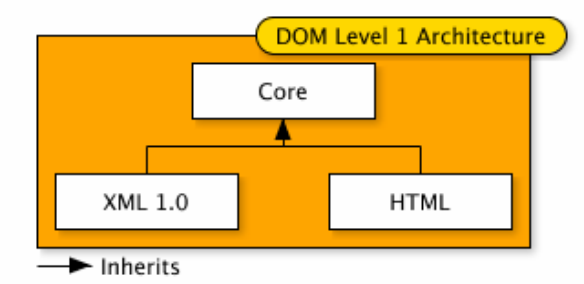
DOM Dokumentenmodell

- das DOM Modell beschreibt
  - logische Struktur eines Dokumentes
  - Zugriff auf Teile eines Dokumentes
  - Manipulation eines Dokumentes
- *core* definiert allgemeine Eigenschaften
  - 12 Objekttypen (u.U. abgeleitete Typen)
- HTML und XML bilden separate Einheiten
  - Abbildung von HTML Elementen auf Objekte
  - u.U. Methoden (z.B. *InsertCell* in Tabellenzeilen)

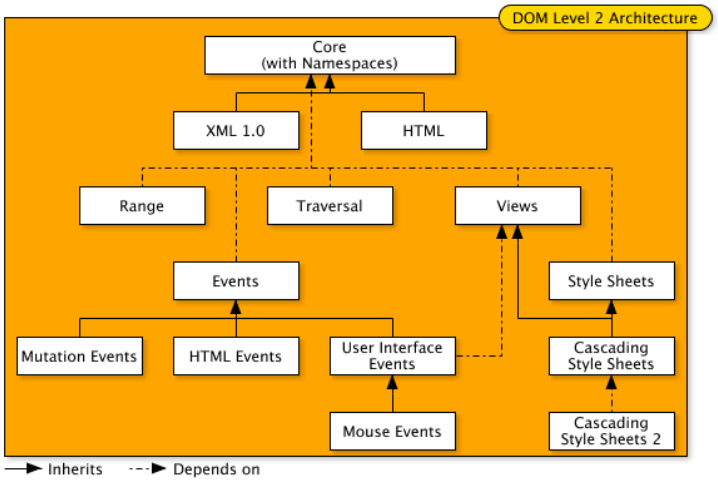
DOM Levels

- DOM Level 0
  - definiert von Netscape 3.0 als JavaScript API
  - keine W3C Spezifikation
- DOM Level 1
  - Oktober 1998, Support für HTML und XML 1.0
- DOM Level 2
  - abgeschlossen November 2000
  - DOM1 + Namespaces, CSS, Events, Traversal
- DOM Level 3
  - momentan in Entwicklung, Alignment mit Infoset

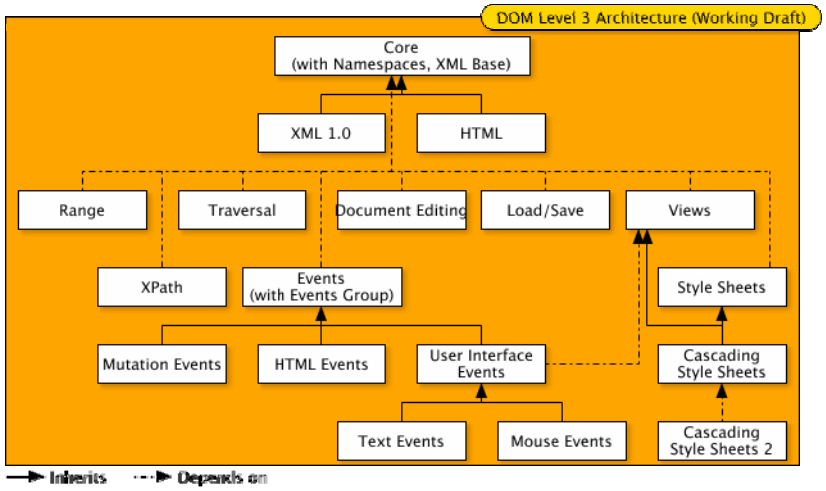
DOM1 Modulhierarchie



DOM2 Modulhierarchie



DOM3 Modulhierarchie



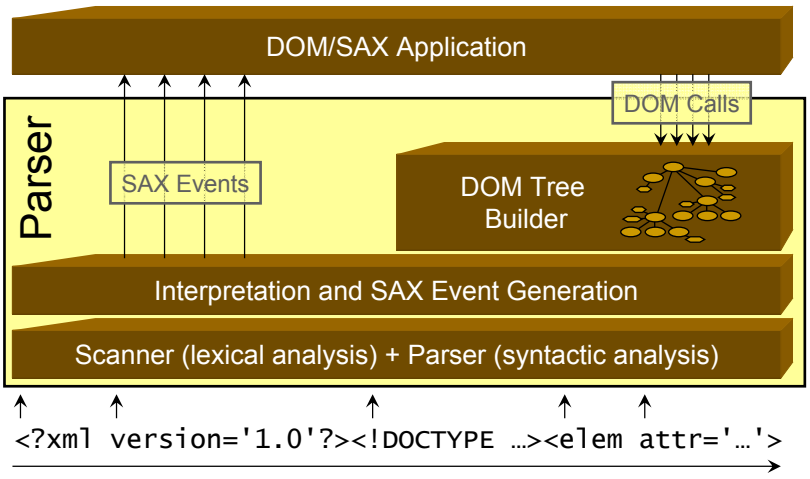
DOM vs. Simple API for XML (SAX)

- Standardisierungsweg
  - DOM wird vom W3C entwickelt (definierter Weg)
  - SAX wird von einer informellen Gruppe definiert
- DOM bietet eine Datenstruktur
  - komplexer und leistungsfähiger
- SAX bietet eine Sequenz von Events
  - einfacher, aber limitiert (praktisch kein Kontext)
- viele Produkte unterstützen DOM und SAX
  - SAX zum Lesen von Daten und Erzeugen von DOM
  - DOM traversieren und SAX Events erzeugen

Simple API for XML (SAX)

- eventbasiertes Interface zu XML
  - gut geeignet für grosse Dokumente
  - gut geeignet für sequentielle Bearbeitung
  - weniger geeignet für komplexe Bearbeitung
- Parser generiert Ereignisse für
  - Start und Ende von Elementen inkl. Attribute
  - Kommentare, Zeichenketten, Entities
  - und alles andere, was im Dokument ist...
- z.B. Apache's Xerces: SAX1/2, DOM1/2
  - DOM basiert meist auf SAX-Primitiven

Aufbau eines DOM/SAX-Parsers



DOM vs. Infoset

- DOM ist deutlich älter (existiert seit 1997)
  - existierte sogar vor XML (API für HTML/JavaScript)
  - Infoset erst seit Oktober 2001 stabil
- Infoset unterstützt verschiedene XML-Standards
  - XML, Namespaces, XML Base
- DOM3 wird an das Infoset angeglichen
  - mit Einschränkungen (z.B. Character vs. Text Nodes)
  - neue Attribute für eine Reihe von Interfaces
  - Normalisierung lässt sich konfigurieren
    - z.B. die Behandlung von Entities und CDATA Sections

Zusammenfassung

- XML Namespaces sichern eindeutige Namen
  - Trennung in URI und lokalen Namen
  - definierte Namespaces für offene Standards
- XML als abstraktes Datenmodell
  - XML Infoset und Canonical XML als Normalform
- Zugriff auf die Daten über APIs
  - DOM als Baum-basiertes Interface zu Dokumenten
  - SAX als simplere Variante für Streaming

DOM und SAX Beispiele

- DOM und SAX in Java
  - http://www.cs.helsinki.fi/u/mraento/teaching/xml_s03/exercises/ex4.html
- basierend auf Apache Xerces
  - http://xml.apache.org/xerces2-j/index.html
  - http://sunsite.cnlab-switch.ch/www/mirror/apache/dist/xml/xerces-j/
- Xerces implementiert DOM und SAX
  - http://xml.apache.org/xerces2-j/api.html
- für die Beispiele notwendig
  - Java Source Code `*DemoPrint.java`
  - Xerces JAR Files
  - Java SDK (`javac` und `java`)