

# Einführung in XML

Erik Wilde  
Institut für Technische Informatik und  
Kommunikationsnetze (TIK)  
ETH Zürich  
<http://dret.net/netdret/publications#uni zh04a>



16.8.04

Erik Wilde

1

## Ziel

- XML als Format für strukturierte Daten
- XML Tools als Toolset für strukturierte Daten
- XML als Grundfertigkeit für Datenmanipulation
- XML-Formate für spezifische Anwendungen
- XML-Transformationen für Umwandlungen



16.8.04

Erik Wilde

2

## Programm

- XML und seine Entstehung
  - Gemeinsamkeiten von XML und HTML
  - Unterschiede von XML und HTML
- XPath als XML-Grundlagenwissen
  - "Reguläre Ausdrücke" für XML
- XSLT als XML-Transformationssprache
  - Umwandlung von XML in Zielformate

16.8.04

Erik Wilde

3

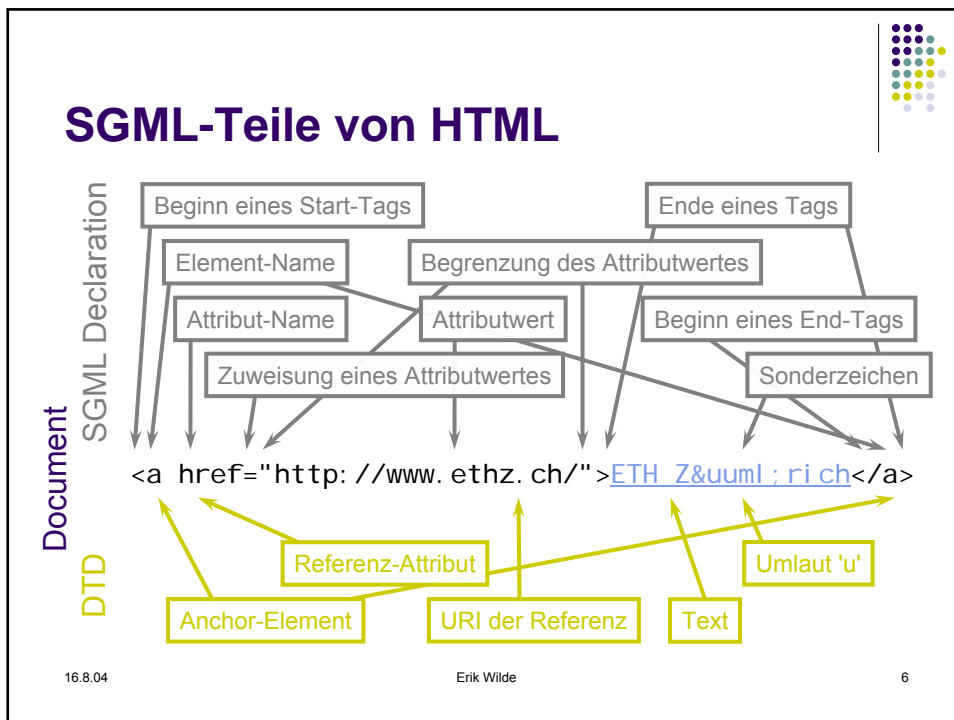
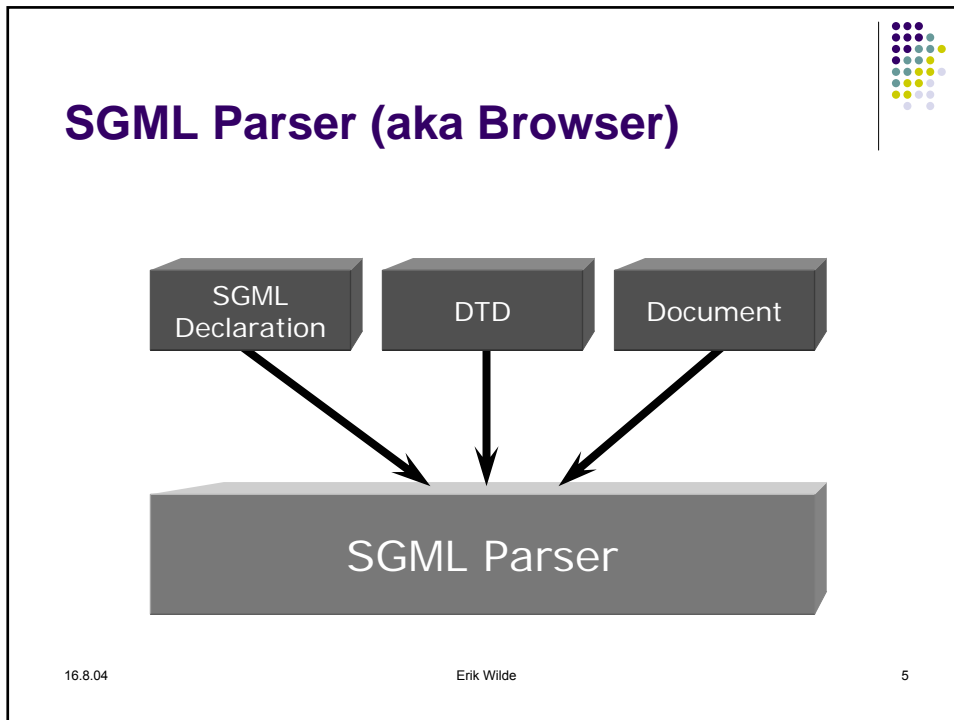
## XML und HTML

- sieht "so ungefähr" aus wie HTML
  - gleiche Basis (*Standard Generalized Markup Language*)
  - *proven success* (SGML und HTML sind Erfolge)
  - geringere Hemmschwelle für Umsteiger
- funktioniert ähnlich wie HTML
  - gleiche Strukturierungsverfahren (Grammatiken)
  - rein textorientiertes Format (keine Binärdaten!)
- andere Zielgruppe als HTML
  - weiterverarbeitbare Information (B2B)
  - anwendungsabhängige Datenstrukturen
  - etabliertes Umfeld (EDIFACT, SGML, proprietär)

16.8.04

Erik Wilde

4



## Vergleich SGML/XML/HTML

	SGML	XML	HTML
<b>SGML Declaration</b>	frei	fix	fix
<b>DTD</b>	frei	frei	fix
<b>Document</b>	frei	frei	frei

16.8.04

Erik Wilde

7

## 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"/>

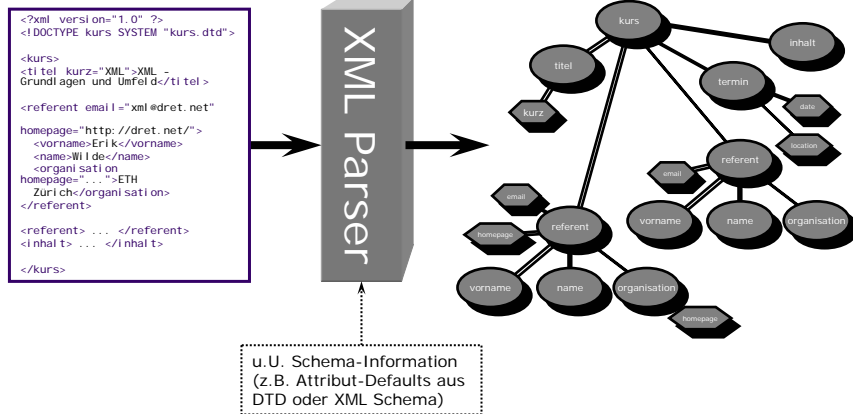
<inhalt> ... </inhalt> </kurs>
```

16.8.04

Erik Wilde

8

## XML Dokumente sind Bäume



16.8.04

Erik Wilde

9

## Document Type Definition (DTD)

- Beschreibung der Datenstrukturen in einem Schema
  - Schema beschreibt eine Klasse von Dokumenten
  - SGML/XML DTD ist nur eine mögliche Variante
- Beschreibung von Datenblöcken
  - Elemente als Strukturmittel
  - Attribute als Daten zu Elementen
- Beschreibung der erlaubten Kombinationen
  - Definition einer Grammatik
  - Verwendung für die Validierung von Daten
- Schema Modellierung als Kern von XML

16.8.04

Erik Wilde

10

## Beispiel (Teil einer DTD)

```

<!ELEMENT kurs          (titel, referent+, termin+, inhalt) >
<!ELEMENT titel         (#PCDATA) >
<!-- ATTLIST titel
      kurz              CDATA #REQUIRED >
<!ELEMENT referent      (vorname, name, organisation?) >
<!-- ATTLIST referent
      email             CDATA #IMPLIED
      homepage         CDATA #IMPLIED >
<!ELEMENT vorname      (#PCDATA) >
<!ELEMENT name         (#PCDATA) >
<!ELEMENT organisation (#PCDATA) >
<!-- ATTLIST organisation
      homepage         CDATA #IMPLIED >

```

16.8.04

Erik Wilde

11

## Elemente

- Elemente sind der grundlegende Mechanismus
  - Strukturierung von hierarchischen Daten
  - "beliebige" Namensgebung für Elemente
  - Definition gemäss inhaltlichen Strukturen
  - Kernpunkt des DTD-Designs
- Elementtypen haben zwei wichtige Aspekte
  - ein *content model* für erlaubten Inhalt
  - Attribute (optionales Vorkommen oder notwendig)
- DTD deklariert Typ, den Dokument verwendet

```

DTD:      <!ELEMENT titel      (#PCDATA) >
Dokument: <titel>XML - Grundlagen und Umfeld</titel>

```

16.8.04

Erik Wilde

12

## Attribute

- Attribute sind Informationen zu Elementen
  - Attribute geben Zusatzinformationen
  - Entscheidung Attribut/Element nicht immer klar
- optional (`#IMPLIED`) oder notwendig (`#REQUIRED`)
- Attribute können verschiedene Typen haben
  - ein Konzept, das für Elemente nicht existiert
  - deutliche Einschränkungen (siehe HTML DTD)

DTD: `<!ATTLIST titel kurz CDATA #REQUIRED >`

Dokument: `<titel kurz="XML">XML - Grundlagen...`

16.8.04

Erik Wilde

13

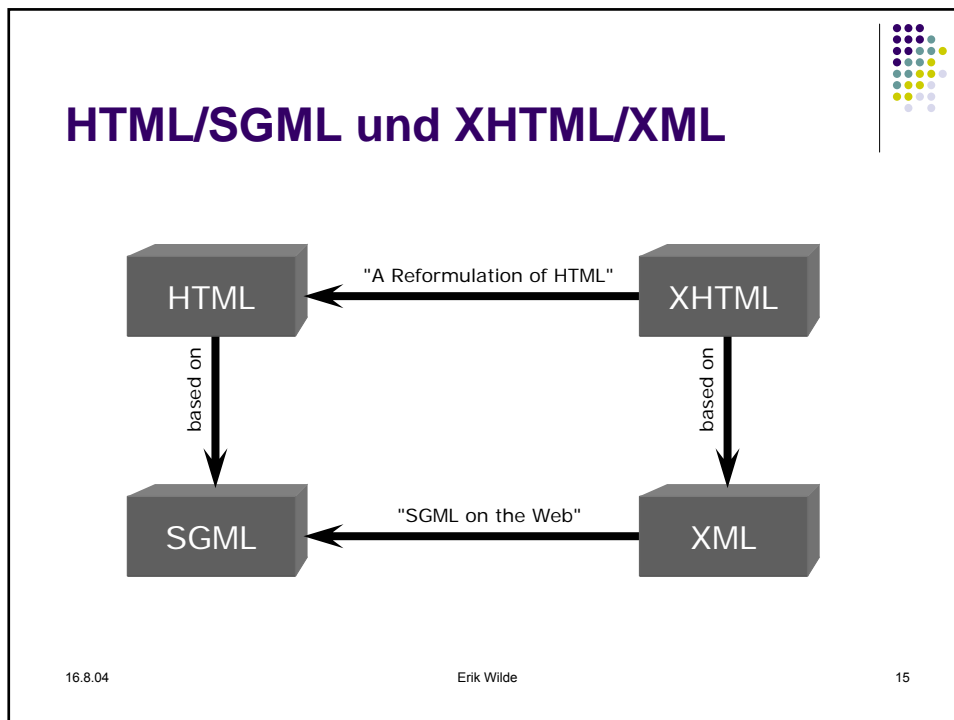
## HTML und XHTML

- HTML basiert auf SGML
  - Gross-/Kleinschreibung ist irrelevant
  - Markup-Minimierung ist erlaubt (Tags weglassen)
- 99% aller HTML-Seiten sind fehlerhaft
  - Tools und Menschen sind nachlässig
  - Browser und Suchmaschinen sind tolerant
  - diese beiden Faktoren verstärken sich gegenseitig
- XHTML ist HTML auf XML-Basis
  - eine XHTML-Seite muss gültiges XML sein
  - damit mit XML-Tools weiterverarbeitbar und erzeugbar

16.8.04

Erik Wilde

14



## HTML → XHTML

- Umwandlung bestehender HTML-Seiten
  1. valides HTML (<http://validator.w3.org>)
  2. alle Elemente und Attribute in lowercase
  3. leere Elemente als `<element />` codieren
  4. End-Tags einfügen (`<p>Text</p>`)
  5. Attributwerte müssen in Quotes sein
  6. vordefinierte Attributwerte müssen lowercase sein
- <http://www.w3.org/TR/xhtml1/#diffs>
- es existieren Utilities zur Umformung
  - <http://tidy.sourceforge.net>

16.8.04 Erik Wilde 16

## Warum XHTML?

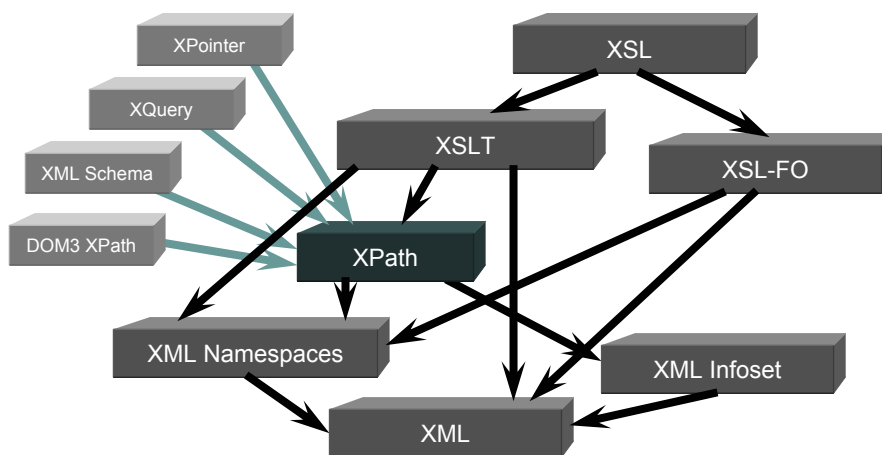
- von allen Browsern unterstützt
  - kann einfach auf dem Server liegen
- kann mit XML Tools verarbeitet werden
  - Reporting, Statistiken, Redesign
- kann mit XML Tools erzeugt werden
  - aus XML-Quellen oder Text-Dateien (XSLT 2.0)

16.8.04

Erik Wilde

17

## XML Path Language (XPath)



16.8.04

Erik Wilde

18

## Bäume: Filesystem vs. XML

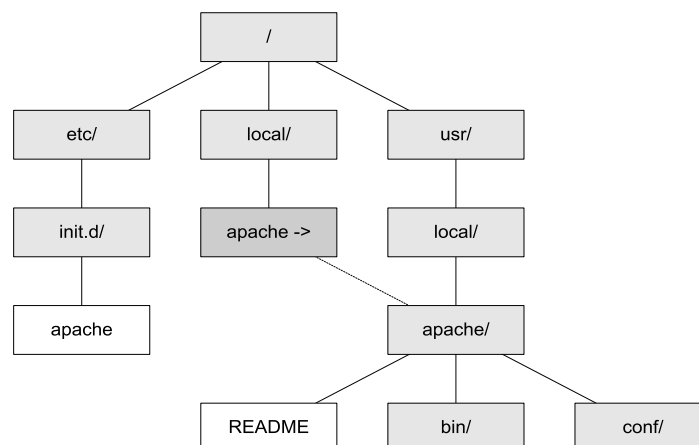
- Unterschiede zwischen der Navigation
    - Unix cd Kommando vs. XPath
  - 1. Knotentypen (*XPath Node Types*)
  - 2. Knotenanzahl (*XPath Node Sets*)
  - 3. Richtungen (*XPath Axes*)
  - 4. Filter (*XPath Predicates*)
- 
- 5. XPath Datenmodell (mehr als nur Knoten)
  - 6. XPath Funktionen

16.8.04

Erik Wilde

19

## Baumsicht eines Filesystems



16.8.04

Erik Wilde

20

## Eigenschaften eines Filesystems



- 3 verschiedene Arten von Knoten
  - Directories, Files, Symbolic Links
- durch das Filesystem garantierte Bedingungen
  - keine Directory-Einträge mit gleichem Namen
  - es gibt immer genau ein Root-Directory
  - Metainformationen werden im Directory gespeichert
    - Owner, Permissions, Creation Date, Modification Date
- verschiedene Sichten des Filesystems
  - *Raw Device* vs. *OS File System Support*
  - entspricht XML Source View vs. XML Tree View

16.8.04

Erik Wilde

21

## 1. Knotentypen



- XPath definiert 7 Knotentypen
  - Root, Element, Attribute, Namespace, Processing Instruction, Comment, Text
  - abgeleitet vom XML Information Set
  - Entstehung des Infosets ist nicht festgelegt
    - XML Parser durch Einlesen eines Dokuments
    - vorangehender Verarbeitungsschritt (z.B. DOM)
- Elemente haben nicht nur Kinder
  - Kinder sind Text, Elemente, PIs und Kommentare
  - Attribute und Namespaces sind keine Kinder
    - aber das Element ist das Elternteil dieser Knoten!

16.8.04

Erik Wilde

22

## 2. Knotenanzahl

- `cd` verlangt immer genau ein Directory
  - dieses wird danach zum *Current Working Directory*
- in vielen Fällen löst die Shell Wildcards auf
  - `ls -laF *.ppt`
  - Kommando muss mehrere Parameter akzeptieren
- spezielle Kommandos erzeugen Listen
  - können per Pipe übermittelt werden
  - `find . -name README -ls | wc`
- Unix hat kein allgemeines Modell für *File Sets*

16.8.04

Erik Wilde

23

## Abarbeitung Filesystem Pfad

- jeder Schritt selektiert genau ein Directory
  - Wildcards im Pfad sind nicht erlaubt
  - Directorynamen sind eindeutig

```
cd /usr/local/apache/bin/
  1 → 1 → 1 → 1 → 1
```

16.8.04

Erik Wilde

24

## Abarbeitung XPath

- jeder Schritt selektiert eine Knotenmenge
  - der nächste Schritt operiert auf dieser Menge
  - pro Schritt kann Menge wachsen oder schrumpfen

/html /body/table/thead/tr

1 → 1 → 1 → 3 → 2 → 4

16.8.04

Erik Wilde

25

## 3. Richtungen

- absolute und relative Pfade
  - Pfad beginnt mit Slash oder mit einem Namen
- Unix Pfade gehen immer einen Schritt abwärts
  - implizite Semantik des Slashes im Pfad
  - . und .. sind keine Ausnahmen, sondern Hacks
    - auf spezielle Directories zeigende Directory-Einträge
- XPath unterstützt verschiedene Richtungen
  - Slash trennt die Schritte (keine implizite Richtung)
  - die Richtung wird durch die XPath Axis angegeben
    - wird sie weggelassen, so ist der Default child

16.8.04

Erik Wilde

26

## XPath Location Path Syntax

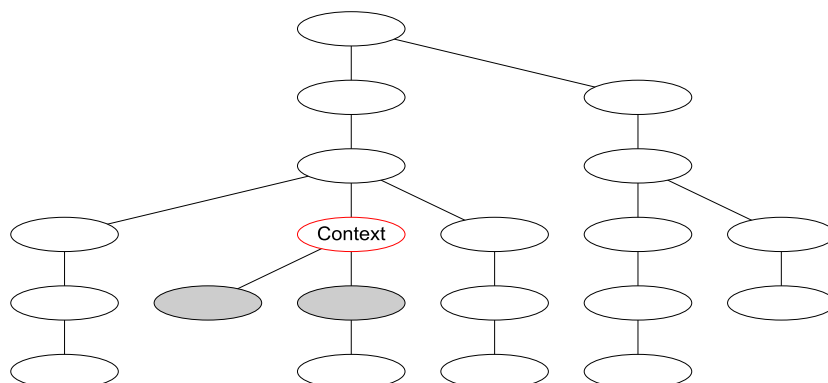
- Folge von *Location Steps*, getrennt mit /
  - erste Komponente: *Axis Specifier* (gefolgt von ::)
  - zweite Komponente: *Node Test*
  - dritte Komponente: 0-n *Predicates* (in [ ])
- Abkürzungen in Location Paths
  - child: : ist die *Default Axis*
  - attribute: : kann abgekürzt werden als @
  - // ist kurz für /descendant-or-self: : node() /
  - . ist kurz für self: : node()
  - .. ist kurz für parent: : node()

16.8.04

Erik Wilde

27

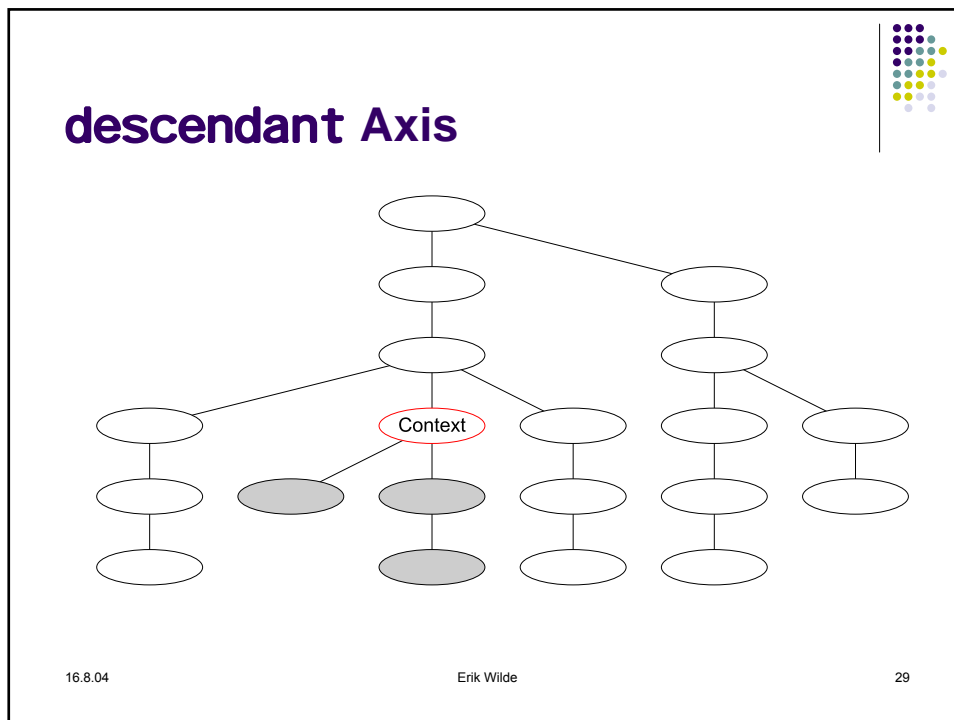
## child Axis



16.8.04

Erik Wilde

28



## XPath Node Tests

- **Namenstest (für Elemente und Attribute)**
  - testet einen Node auf einen bestimmten Namen
    - `/html /head/ tabl e`
  - \* kann verwendet werden als Wildcard Character
- **Test auf den Nodetype**
  - markiert mit nachfolgenden Klammern
  - Test auf `text()`, `comment()` oder `node()`
- **Test auf Processing Instruction**
  - anwendbar auf spezifische Processing Instruction
    - `processi ng-i nstructi on("myappl i cati on")`

16.8.04 Erik Wilde 30

## 4. Filter (Prädikate)

- jeder Schritt kann zusätzliche Filter enthalten
  - Auswertung jedes Filters für jeden selektierten Node
    - daraus resultiert eine logisches *and* der Filter
  - Filter sind XPath Expressions
    - Auswertung relativ zum selektierten Node
    - Interpretation als boolean (ausser für number)
- Reduktion der Resultatmenge eines Schrittes
  - `/descendant::table[not(thead)]`
  - können in einigen Fällen vermieden werden
    - `/descendant::table[thead]`
    - `/descendant::table/thead/..`

16.8.04

Erik Wilde

31

## Filter in XML Bäumen

- Abarbeitung eines Location Paths
  1. Node Set ist die Basis jedes Schrittes
    - Context: Node, Context Size, Context Position, ...
  2. Selektion der Nodes (Axis und Node Test)
  3. Auswertung aller Filter für alle selektierten Nodes
  4. die verbleibenden Nodes sind das neue Node Set
- Aufgabe: Finden Sie den 6. Link im HTML
  - Ansatz 1: `//a[position()=6] ≡ //a[6]`
    - Problem: `/descendant-or-self::node()/a[6]`
  - korrekter XPath: `/descendant::a[6]`

16.8.04

Erik Wilde

32

## 5. XPath Datenmodell

- XPath unterscheidet 4 Datentypen
  1. node-set (Mengen von Knoten)
  2. boolean (true oder false)
  3. number (Floating Point Numbers)
  4. string (Unicode Character strings)
- Beispiele für XPaths dieser Datentypen
  1. //table[thead]: alle Tabellen mit einem thead
  2. 1=2: ergibt den Boolean Wert false
  3. 27 div 16: ergibt 1.6875
  4. substring('abcd', 2, 2): ergibt den String bc

16.8.04

Erik Wilde

33

## XPath Umgang mit Datentypen

- XPath definiert viele implizite Konvertierungen
  - keine gute Grundlage für eine typensichere Sprache
  - Umformungsregel müssen bekannt sein
    - es gibt explizite Funktionen, die diese Regeln definieren
    - diese werden u.U. implizit angewandt
- viele XPath sind "korrekt"
  - häufiger Effekt beim Lernen von XPath
    1. Typ-Probleme (oder anderes...) im XPath
    2. Resultat ist ein leeres Node Set
    3. kann als Boolean, String oder Number interpretiert werden

16.8.04

Erik Wilde

34

## 6. XPath Funktionen



- Funktionen mit dem Resultattyp `boolean`
  - `boolean`, `contains`, `false`, `lang`, `not`, `starts-with`, `true`
- Funktionen mit dem Resultattyp `number`
  - `ceiling`, `count`, `floor`, `last`, `number`, `position`, `round`, `string-length`, `sum`
- Funktionen mit dem Resultattyp `string`
  - `concat`, `local-name`, `name`, `namespace-uri`, `normalize-space`, `string`, `substring`, `substring-after`, `substring-before`, `translate`
- Funktionen mit dem Resultattyp `node-set`
  - `id`

16.8.04

Erik Wilde

35

## Location Paths und Funktionen

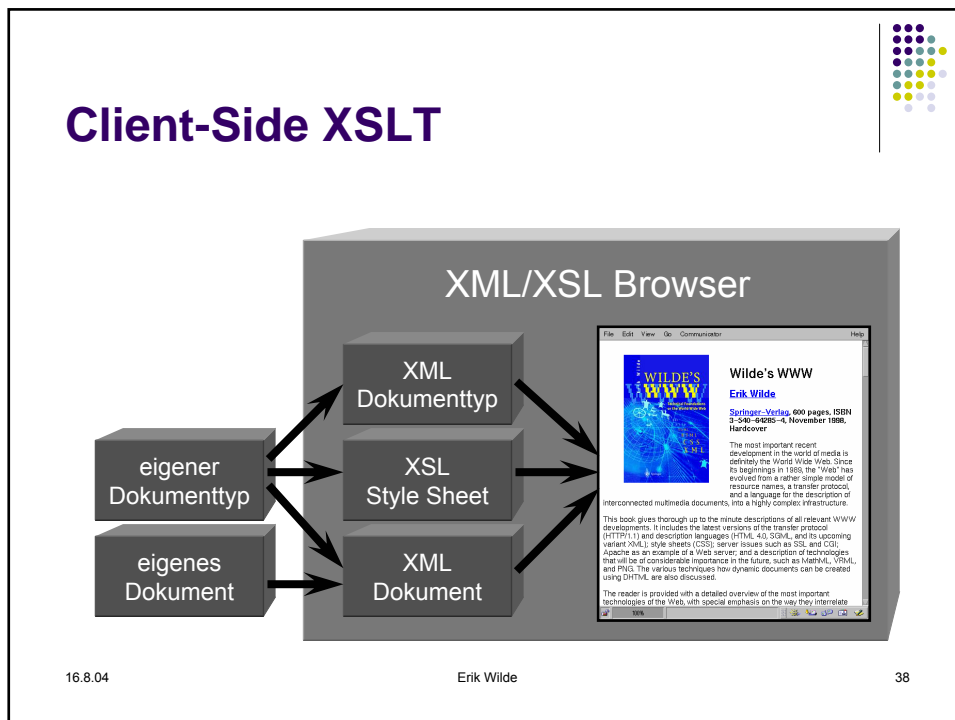
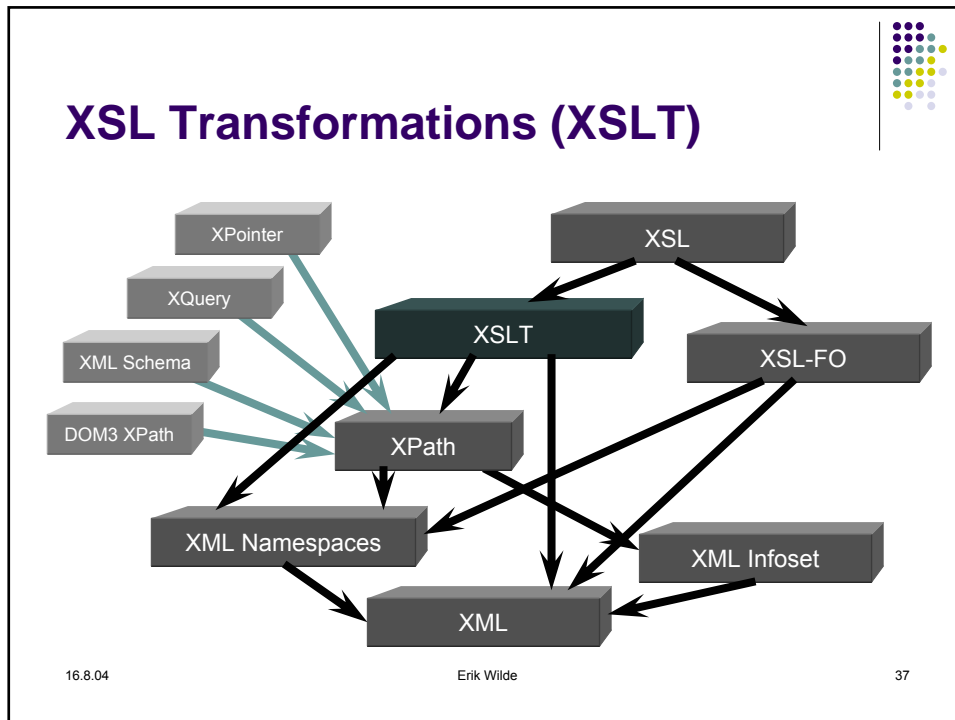


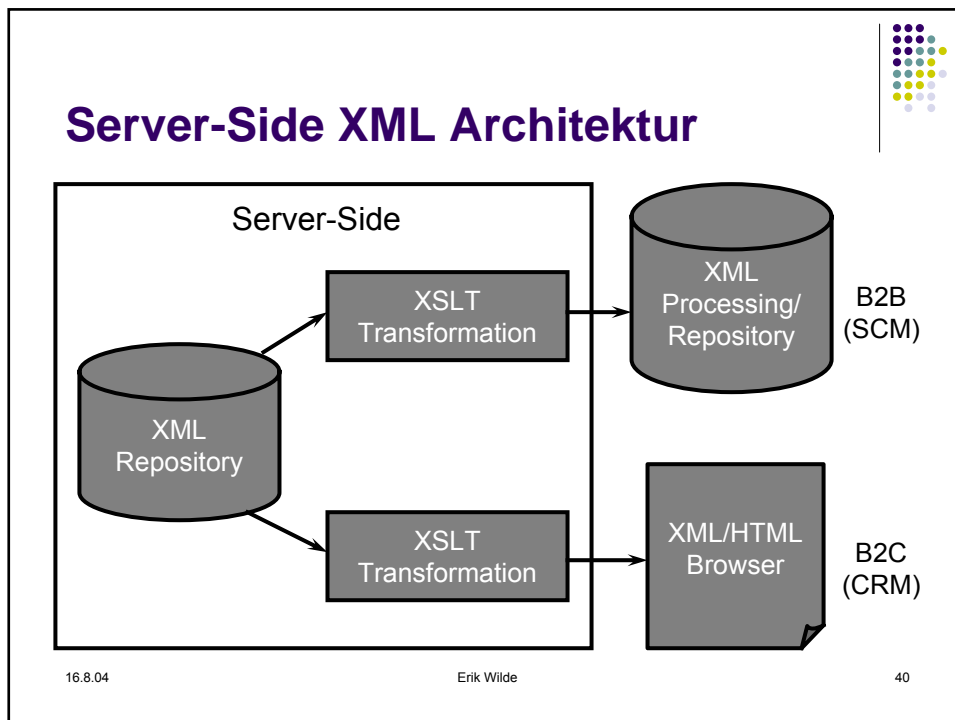
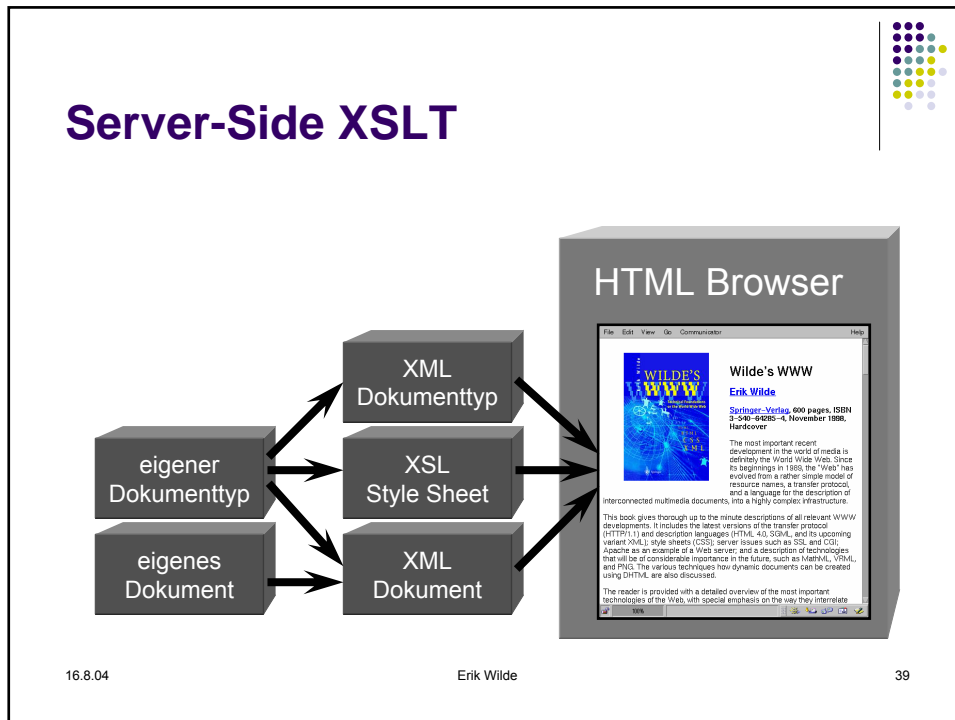
- Location Paths können Funktionen benutzen
  - kommen dann immer in Prädikaten vor
  - `//a[substring(@href, string-length(@href)-2)='pdf']`
- Funktionen können Location Paths benutzen
  - falls die Funktion Node Sets akzeptiert
    - `count(//a)` zählt die Anzahl der Links auf einer Seite
    - andernfalls findet eine implizite Konvertierung statt
- dieses Prinzip kann geschachtelt werden
  - die Übersichtlichkeit kann leiden
    - `//*[ @* ]` oder `//*[ count( @* ) = count( * ) ]`
  - sehr komplexe Aufgabenstellungen lösbar

16.8.04

Erik Wilde

36





## Transformation von XML



- XML definiert Dokumententypen
  - typischerweise Szenarien mit vorgegebenen DTDs
  - selbst definierte DTD (Verwendung nur in-house)
  - vordefinierte DTD (gegeben von externer Instanz)
    - für Präsentation oder Austausch notwendig
- Grundidee der Transformationen von XML
  - zuerst aufgetaucht im *Style Sheet* Ansatz
  - Transformation in Strukturen eines anderen Schemas

16.8.04

Erik Wilde

41

## XSL Transformations (XSLT)

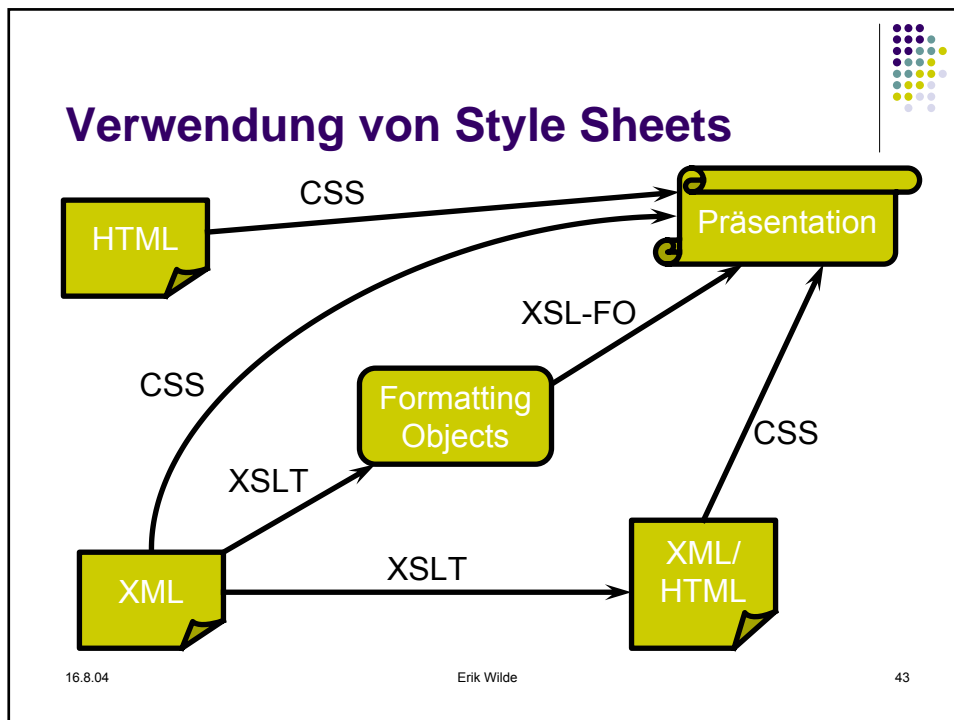


- ursprünglich Teil des XSL Standards
  - jetzt per Referenz in den Standard integriert
  - orientiert an Scheme, einem Lisp-Dialekt
- Potential wurde erkannt
  - Trennung von XSL in XSLT und XSL-FO
  - XSLT transformiert in *XSL Formatting Objects*
  - XSL-FO werden zur Darstellung benutzt
- XSLT wurde weiter geteilt
  - XPath für die Selektion von Teilen eines Dokuments
  - "der Rest", die Kontrollstrukturen
- besserer Name: *XML Transformation Language*

16.8.04

Erik Wilde

42



## XSLT: Hello World!

```

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

  <xsl:template match="/">
    <xsl:text>Hello World! </xsl:text>
  </xsl:template>

</xsl:stylesheet>
  
```

- Abarbeitung beginnt beim Root Node
- Ausgabe eines Text Nodes
- Ende der Abarbeitung

16.8.04 Erik Wilde 44

## XSLT Beispiel

```

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

  <xsl:output method="text"/>

  <xsl:template match="*">
    ( Element: <xsl:value-of select="local-name()"/>
      <xsl:apply-templates select="* | @*" /> )
  </xsl:template>

  <xsl:template match="@*">
    Attribute: <xsl:value-of select="local-name()"/>
  </xsl:template>

</xsl:stylesheet>

```

16.8.04

Erik Wilde

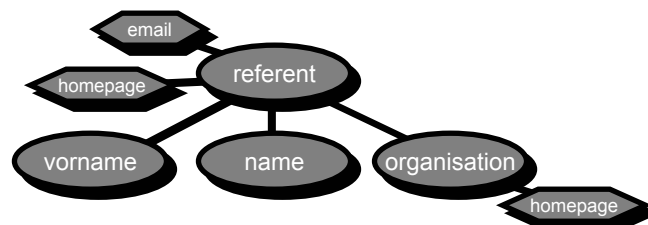
45

## Eingabedokument

```

<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>

```



16.8.04

Erik Wilde

46

## Resultat der Ausführung

```
( Element: referent
  Attribute: email
  Attribute: homepage
  ( Element: vorname )
  ( Element: name )
  ( Element: organization
    Attribute: homepage ) )
```

- Whitespace nachträglich von Hand verändert!

16.8.04

Erik Wilde

47

## Beispiel unter der Lupe

- Steuerung der Ausgabe möglich
  - normalerweise XML als Ausgabe
  - Text-orientierte Ausgabe ebenfalls erlaubt
- Programmsteuerung durch *Template Rules*
  - Rekursion als Normalfall
  - Selektion der Template Rules durch XSLT Prozessor
  - Ausführung durch das Dokument gesteuert
- inkrementelle Entwicklung
  - ohne die zweite Template Rule ebenfalls lauffähig
  - aber: Default-Verhalten in diesem Fall ungünstig

16.8.04

Erik Wilde

48

## XSLT aus der Ferne betrachtet



- Eingabe ist ein XML-Dokument
  - etwas genauer betrachtet ein XPath Node Tree
    - basiert auf dem Infoset
    - oftmals *Whitespace Stripping* vor der Transformation
- Transformation als Ausführung des XSLT
  - beliebige Komplexität der Abarbeitung
- Ausgabe ist XML, HTML oder Text
  - XML ist der Normalfall (erlaubt Konkatination)
  - HTML als populäres Präsentationsformat
  - Text ohne Markup-Struktur

16.8.04

Erik Wilde

49

## Besten Dank



Fragen? Kommentare?

mailto:net.dret@dret.net

Homepage: <http://dret.net/netdret/>

Online Glossar: <http://dret.net/glossary/>

16.8.04

Erik Wilde

50