XML Schemasprachen

Übersicht und Einordnung

Erik Wilde (ETH Zürich)

Übersicht

- DTDs und XML Schema
  - Einschränkungen und Nachteile
- Schemasprachen als Konzept
  - Grammatiken und Anderes
  - ISO Document Schema Definition Languages
- Schemasprachen selbstgemacht
  - wann eigene Schemasprachen sinnvoll sind
  - Implementierung eigener Schemasprachen

XML DTDs

- die XML Spezifikation definiert zwei Syntaxen
  - die Syntax von XML Dokumenten
  - die Syntax von XML DTDs
- DTDs definieren Grammatiken
  - Aufbauregeln für XML Dokumente
  - definieren unendlich grosse Mengen an Dokumenten
- DTDs müssen von Parsern interpretiert werden
  - Entity-Definitionen sind in der DTD

Inhalt einer DTD

- Elementtyp Definitionen
  - Element und sein Content Model
- Attributlisten Definitionen
  - Mengen von Attributen für Elemente
  - der Typ von Attributen (u.U. Aufzählungsliste)
  - diverse Qualifier für Attribute (optional, Default, ...)
- Entity Definitionen
- XML Parser müssen all das unterstützen

Nachteile der DTDs

- keine Beziehungen zwischen Elementtypen
  - keine Typ-Hierarchie der Elemente
- keine Unterstützung von Wiederverwendung
  - verbreitetes Parameter Entity Design Pattern
- keine anwendungsorientierten Datentypen
- keine Unterstützung für XML Namespaces
  - "DTDs and Namespaces don't mix"
- keine XML Syntax
  - kann nicht mit XML Tools verarbeitet werden

XML Schema

- Kompromiss aus vielen Vorschlägen
  - umfangreiche Bibliothek an eingebauten Datentypen
- Modellierungshierarchien
  - Typenhierarchien für Simple und Complex Types
    - basierend auf einem konyeptionellen *Ur-Type*
  - unterschiedliche Ableitungsmethoden
    - Simple: Restriction, List und Union
    - Complex: Restriction und Extension
- Ziel war eine möglichst mächtige Sprache
  - nicht unbedingt der beste Ansatz (monolithisch)

XML Schema Eigenheiten

- keine Unterstützung von Entities
  - falls benötigt, in einer minimal-DTD definiert
- Identity Constraints als `ID/IDREF++`
  - basieren auf Elementnamen (keine Typen!)
- ist definiert auf Infosets
  - keine Validierung von Dokumenten
  - Validierung ergänzt das Infoset
  - Zugriff nicht durch einen Standard definiert
    - Abhängigkeit von der spezifischen Implementierung

Nachteile von XML Schema

- hohe Komplexität
- momentan nur mangelhafte Implementierungen
  - zum Schreiben von XML Schema selber
    - einzig IBM's SQC implementiert XML Schema komplett
  - zum Validieren von XML Dokumenten
    - keine komplette Implementierung bekannt
- keine Co-Constraints
- Zeichen in *mixed Content* haben keinen Typ
  - keine Einschränkungen für *mixed Content* möglich

Designing XML Schemas

- Evolution von XML Schemas
  - neue Versionen von Schemas (2.0 vs. 1.0)
  - alte Software mit neues Instanzen
- es gibt viele Erweiterungsmethoden
  - Type Derivation (Simple und Complex)
  - Substitution Groups
  - Type Substitution (Type Casting in der Instanz)
- Designing for Change
  - die meisten Dinge sind per Default erlaubt

Verbindung Instanz/Schema

- verschiedene Mechanismen sind denkbar
  - irgendwie verankert in der XML Syntax
- DTDs benutzen spezielle Konstrukte
  - Document Type Declaration: `<!DOCTYPE ...`
- XML Schema benutzt Namespaces
  - es gibt einen Instance Namespace
- RELAX NG definiert keine Verbindung
- XML liefert weitere Varianten
  - Kommentare, Processing Instructions, Namespaces

RELAX NG

- alternative Grammatik-basierte Schemasprache
- basierend auf bekanntem Formalismus
  - DTDs benutzen einen ad hoc Formalismus
  - XML Schema benutzt einen ad hoc Formalismus
  - RELAX NG benutzt *Hedge Automata*
- keine Typen (⇒ keine Typhierarchien)
- keine eigene Typbibliothek
  - verwendet häufig die XML Schema Simple Types

RELAX NG Prinzipien (I)

- das Dokument bleibt unverändert
  - keine Default-Werte im Schema
  - keine Entities
- nur Beziehungen Instanz-Schema
  - DTDs und XML Schema: Instanz-Instanz
    - `ID/IDREF(S)` bzw. Identity Constraints
  - XML Schema: Schema-Schema
    - Typhierarchie

RELAX NG Prinzipien (II)

- Gleichbehandlung von Attributen und Elementen
  - Content Model enthalten beide
  - ermöglicht viele nützliche Anwendungen
- Verbesserung der ANY Group
  - Verallgemeinerung von DTD und XML Schema
  - Anforderung aus dem 'Dokument'-Bereich

DTD Beispiel

```
<!ELEMENT document (heading, chapter)>

<!ELEMENT heading (#PCDATA)>

<!ELEMENT chapter (heading, para+)>

<!ELEMENT para (#PCDATA)>
```

XML Schema Beispiel

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="document">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="heading"/>
    <xs:element ref="chapter"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="heading" type="xs:string"/>
 <xs:element name="chapter">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="heading"/>
    <xs:element name="para" type="xs:string" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```


XML Schema Beispiel (XSCS)

```
element document {
   ( heading, chapter )
}

element heading { xs:string }

element chapter {
   ( heading, paragraph { xs:string }+ )
}
```

RELAX NG Beispiel

```
<grammar xmlns="http://relaxng.org/ns/structure/1.0">
 <start><ref name="document"/></start>
 <define name="document">
  <element name="document">
   <ref name="heading"/>
   <ref name="chapter"/>
  </element>
 </define>
 <define name="heading">
  <element name="heading"><text/></element>
 </define>
 <define name="chapter">
  <element name="chapter">
   <ref name="heading"/>
   <oneOrMore>
    <element name="para"><text/></element>
   </oneOrMore>
  </element>
 </define>
</grammar>
```

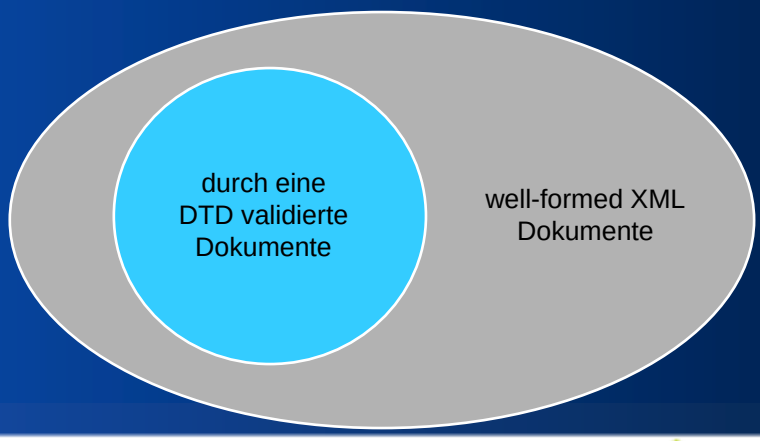
RELAX NG Beispiel (Compact Syntax)

```
start = document

document = element document { heading, chapter }

heading = element heading { text }

chapter = element chapter {
    heading,
    element para { text }+
  }
```

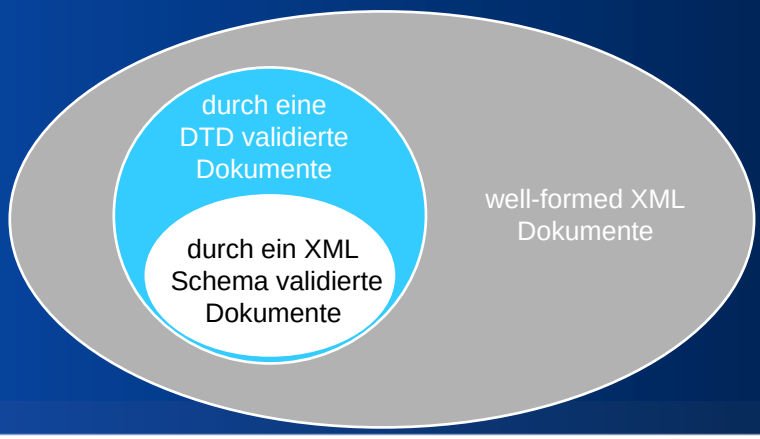
Grenzen von Schemasprachen

- was kennen Schemasprachen?
  - das Schema und die Instanz
  - u.U. in die Schemasprache eingebaute Dinge
    - Typbibliotheken, Funktionen, ...
- was kennen sie nicht?
  - Daten ausserhalb von Schema oder Instanz
- Schemasprachen können sehr einfach sein
  - `(/order/@shipdate > today() + 2) & (/order/@cid = dblookup(customer))`

Schemasprachen als Konzept



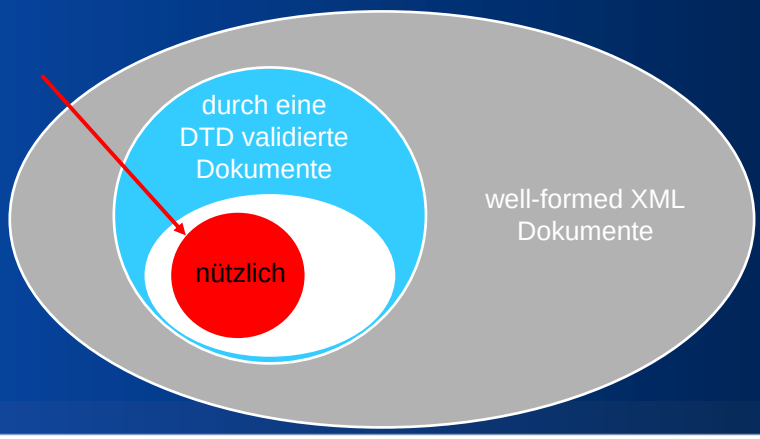
XML Schema als DTD++



Wozu überhaupt Schemasprachen?

- Einschränkung der akzeptierten Dokumente
  - Verwendung von Standardsoftware zur Validierung
- Schemasprachen können nie alles
  - DTDs unterstützen kaum Datentypen
    - dass ein Attributwert eine positive Zahl sein muss
  - XML Schema unterstützt keine Co-Constraints
    - dass ein Attributwert kleiner sein muss als ein anderer
  - Schematron kann nicht mit externen Daten arbeiten
    - dass ein Attributwert in einer Datenbank existieren muss

Schema-valid vs. Application-valid



Schemasprachen sparen Geld

- Schemas schränken XML deklarativ ein
  - häufig verwendete Software-Komponenten
    - tendenziell weit verbreitet und ausgereift
- keine Eigenentwicklung von Software nötig
  - falls Software für die Schemasprache existiert
  - einfacher Update (neuen Validator installieren)
- besserer "Schutz" der Applikation
  - weniger Fehlerbehandlung in der Applikation

Applikationsanforderungen

- Schemasprachen sind nie komplett
  - Code ist immer notwendig
  - das Ziel ist so wenig Code wie möglich
- zwei prinzipielle Wege sind möglich
  - Schemasprachen kombinieren
  - Schemasprachen erweitern oder definieren
  - beide Wege können kombiniert werden
- gute Planung spart Implementierungsaufwand

Klassifizerung von Schemasprachen

- Grammatik-basierte Sprachen
  - sehr weit verbreitet in der XML-Welt (DTD)
  - DTD, XML Schema, RELAX NG
- regelbasierte Sprachen
  - definieren Regeln, die eingehalten werden müssen
  - oftmals gute Ergänzung zu Grammatiken
- andere Sprachen
  - momentan noch stark in Entwicklung

Schematron

- benutzt *Tree Patterns*
  - Identifikation von Teilen (mittels XPath)
  - Tests basierend auf dem selektierten Context
- sehr einfache Abbildung auf XSLT
- zwei Arten von Constraints
  - Assertions müssen gelten (`true` ergeben)
    - anderfalls wird eine Aktion ausgelöst
  - Reports informieren über Tatsachen
    - werden ausgegeben, falls sie `true` ergeben

Schematron Beispiel

```
<house>
 <wall/><wall/><wall/><wall/>
 <window/><window/>
 <roof/>
</house>

<rule context="house">
 <assert test="count(wall) = 4">A house should have four
   walls</assert>
 <report test="not(roof)">This house does not have a
   roof</assert>
 <report test="window">This house has windows</assert>
</rule>
```

Schematron Eigenschaften

- Rules definieren den Context von Tests
  - enthalten Tests (Assertions und Reports)
- Abstract Rules ermöglichen Modularität
  - Referenzierung aus anderen Rules
- Pattern fassen Rules zusammen
- Phases referenzieren Patterns
  - Anwendung des Schemas zu verschiedenen Zeiten
  - Parametrisierung der Validierung

Schemasprachen selbstgemacht

- wechselnde Anforderungen einer Applikation
  - z.B. verschiedene Anwendungskontexte
  - z.B. Anbindung an unterschiedliche Infrastruktur
- Schemasprachen können einfach sein
  - Abbildung auf bestehende Software-Komponenten
- Aufwand vs. Ertrag
  - einfache Parametrisierung einer XML-Anwendung
  - Selbstschutz der Entwickler (Encapsulation)

Zusammenfassung

- Schemasprachen als Programmierung
  - deklarativ vs. prozedural
  - u.U. eigene Schemasprachen benutzen
- wachsende Zahl an Schemasprachen
  - Toolset an klar fokussierten Tools
  - modularer Aufbau einer Anwendung
- Validierung als Pipeline
  - Kombination verschiedener Schemasprachen

Q&A

Danke für Ihre Aufmerksamkeit!

Erik Wilde (ETH Zürich)