



# XML Schema

Erik Wilde  
ETH Zürich  
<http://dret.net/netdret/>



## Abstract

*XML Schema* wird als Grundlage für eine zunehmende Anzahl von XML-Technologien (Web Services, XQuery, XSLT 2.0) immer wichtiger. XML Schema führt als wichtigste Neuerung eine Typebene in die Welt von XML ein, die aus zwei grundlegenden Arten von Typen (*Simple* und *Complex Types*) besteht. Dieses Typkonzept und die Konsequenzen für XML-Anwendungen werden in dieser Session vorgestellt.

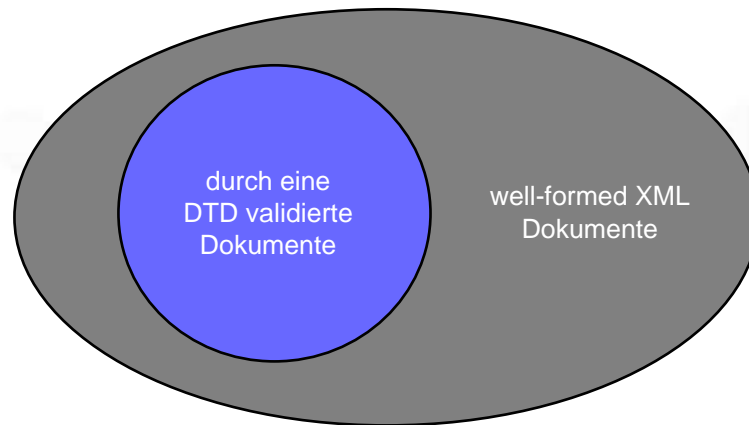
## Übersicht

- DTDs → XML Schema
- Anwendungen von XML Schema
- Eigenschaften von XML Schema
  - Typen und ihre Verwendung
  - lokale und globale Deklarationen
- Umgang mit XML Schema
  - Modellierung und Programmierung
- Zusammenfassung

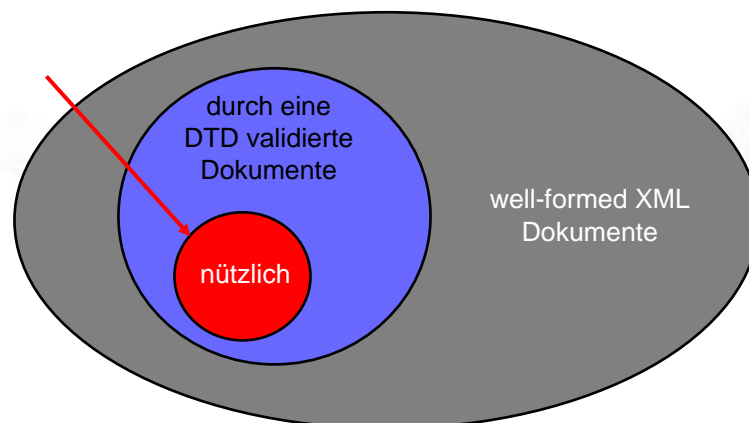
## Nachteile von DTDs

- keine Beziehungen zwischen Elementtypen
  - keine Typ-Hierarchie der Elemente
  - zusammenhangsloses Nebeneinander
- 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

## Schemasprachen als Konzept



## DTD-valid vs. Application-valid



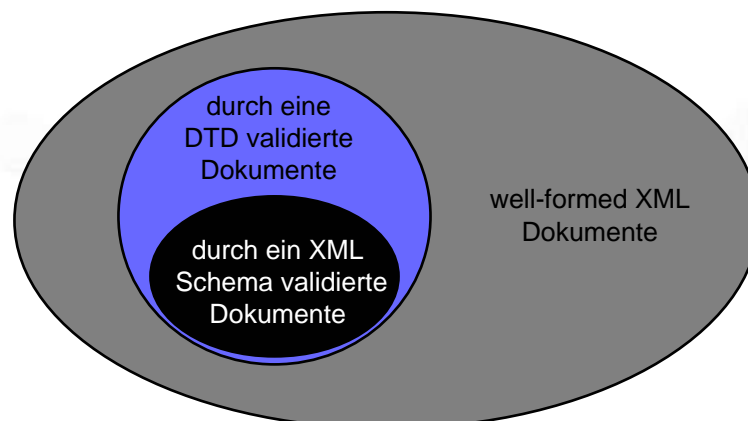
## XML Schema

CLIX · CRVX · DDML · DSD · DT4DTD · DTD ·  
Examplotron · MNS · MSL · NRL · RELAX ·  
RELAX NG · SAF · Schematron · SOX · TREX ·  
XCSL · XDR · XML-Data · XML Schema

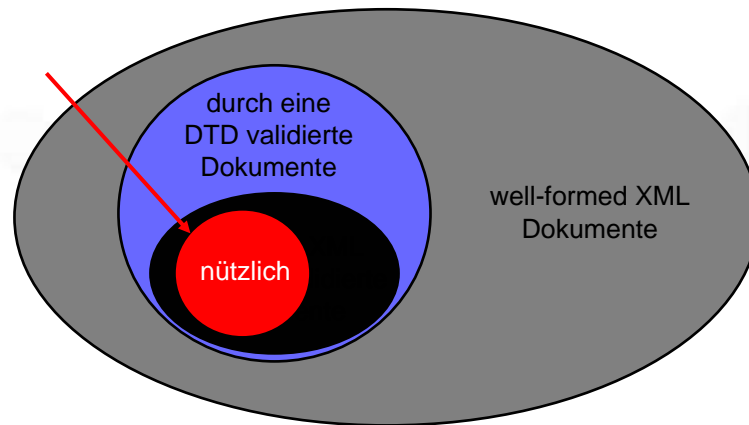
<http://dret.net/glossary/xml-schemalanguage>

- viele verschiedene Verbesserungsvorschläge
  - DTD um Datentypen erweitern (DT4DTD)
  - weitergehende Ansätze verschiedener Akteure
    - XML Data von Microsoft, später reduziert auf XDR
  - Konzentration der Aktivitäten beim W3C
    - alle wichtigen Akteure waren beteiligt
- separate Spezifikation des W3C
  - keine Veränderung der XML Spezifikation
  - basiert auf verschiedenen W3C Spezifikationen
    - XML 1.0, XML Infoset, XML Namespaces, XML Base
  - mittlerweile Basis vieler XML-Technologien

## XML Schema als DTD++



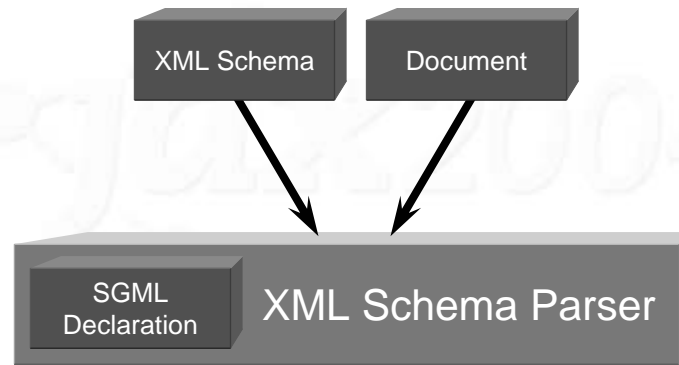
## Schema-valid vs. Application-valid



## Valid und schema-valid XML

- XML unterscheidet zwischen zwei "Levels"
  - *well-formed* gehorchen dem XML-Standard
  - *valid* sind well-formed und gehorchen einer DTD
- well-formed und valid Konzepte
  - sind direkt im XML Standard definiert
  - können mit DTD und Dokument verifiziert werden
- *schema-valid* Dokumente
  - müssen gemäss einem XML Schema validiert werden
  - gibt es nur mit XML Schema Applikationen
  - haben mehr Randbedingungen als valid Dokumente

## XML Schema Parser



## XML Schema validiert Bäume

- streng genommen keine XML-Anwendung
  - auf dem Infoset (XML Abstraktion) definiert
- Validierung ist definiert als Infoset-Operation
  - Eingabe ist ein XML Infoset
  - die Validierung ergänzt das Infoset um Informationen
    - *Post-Schema Validation Infoset (PSVI) Contributions*
  - Ausgabe ist ein annotiertes Infoset
- viel verschiedene Information im PSVI
  - Erfolg der Validierung (*valid/invalid/partial/skipped*)
  - Typ-Informationen (kommen nicht aus der Instanz)
    - Verweis auf die *Schema Components*

## XML Schema & Web Services

- Web Services sind XML-basierte Applikationen
  - Definition der Schnittstellen mittels *WSDL*
    - Typdefinitionen in WSDL benutzen XML Schema
  - Austausch der Daten mittels *SOAP*
    - Instanzen von XML Schema definierten Schnittstellen
- Modellierung von WSDL Datentypen
  - im einfachsten Fall Generierung aus einer Instanz
    - schnell und einfach, aber kein "gutes" XML Schema
  - für langlebige komplexe Anwendungen: gute Modelle
    - Versionierung der WSDL Schnittstellen
    - u.U. Koexistenz verschiedener Softwareversionen

## XML Schema & XQuery

- XQuery wird die Abfragesprache für XML
  - in der ersten Version keine Manipulation von Daten
  - zum grossen Teil auf XPath 2.0 beruhend
- Abfragesprachen brauchen Datentypen
  - ansonsten gibt es nur textuelle Vergleiche
    - untauglich für Zahlen, Daten, Zeiten, ...
  - ansonsten gibt es keine Relationen
    - grösser, kleiner, Intervalle, ...
- XQuery benutzt die XML Schema Datentypen
  - an sich benutzt XQuery das XPath 2.0 Datenmodell

## XML Schema & XSLT 2.0

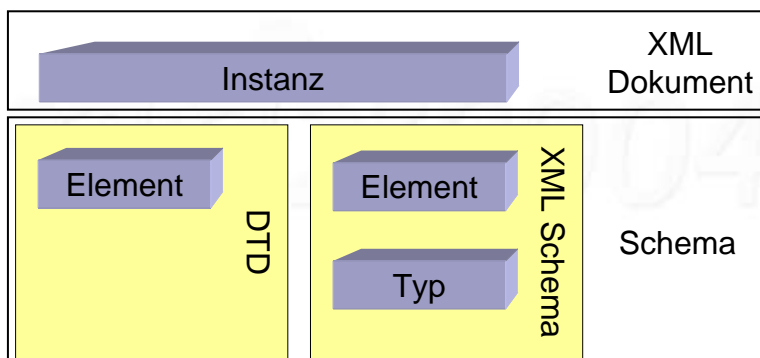
- XSLT 2.0 erweitert XSLT 1.0 stark
  - deutliche Verbesserung des Sprachumfangs
  - deutliche Verbesserung der Expression Language
- XSLT 1.0 hat ein sehr einfaches Datenmodell
  - XPath 1.0: Node Sets, Boolean, Number, String
  - keine weitergehenden Datentypen
    - Daten, Zeiten, Zeitintervalle, ...
    - keine Operationen auf solchen Werten möglich
- XSLT 2.0 benutzt die XML Schema Datentypen
  - an sich benutzt XSLT 2.0 das XPath 2.0 Datenmodell

## Was ist neu an XML Schema?

- XML Schema führt Datentypen ein
  - Datentypkonzept orthogonal zu XML
    - soweit es eben geht, XML 1.0 bleibt unverändert
  - *Simple Types* und *Complex Types*
- XML Schema führt Typableitungen ein
  - verschiedene Arten der Typableitung sind möglich
  - es ergibt sich eine Typhierarchie
- XML Schema modernisiert I D/I DREF
  - (fast) alles wird besser als in DTDs
  - aber der I DREFS Typ wird nicht mehr unterstützt



## XML Schema Typen



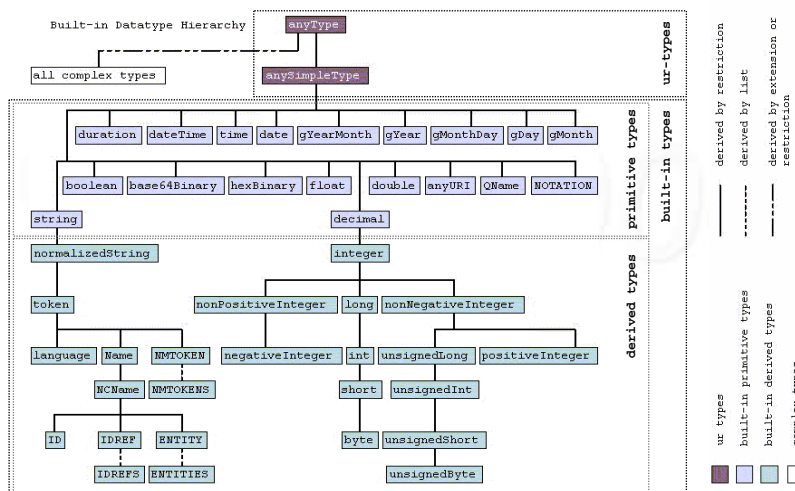
## Was ist ein Attribut?

- was ist ein Attribut in der Instanz?
  - was XML 1.0 definiert wie ein Attribut aussieht
  - zusätzlich dazu die Definition der XML Namespaces
- wie wird ein Attribut definiert?
  - ein Attribut hat einen *Simple Type*
  - es kann optional oder verpflichtend sein
  - es kann einen Defaultwert haben
  - es kann an drei Stellen definiert werden
    1. als Attribut eines *Complex Type*
    2. als eigenständiges Attribut zur Referenzierung
    3. innerhalb (u.U. geschachtelter) Gruppen von Attributen

## Was ist ein Simple Type?

- definiert die Grundbausteine von
  - XML Schema
  - und damit auch XML Dokumenten
- definiert Inhalt von Elementen oder Attributen
  - Element: `<i sbn>0130655678</i sbn>`
  - Attribut: `<buch i sbn="0130655678">`
- drei Varianten von Simple Types
  1. Atomic Types (kleinste Einheit, z.B. Zahlen)
  2. List Types (mit Space getrennt, z.B. "3 5 7")
  3. Union Types (Vereinigung anderer Simple Types)

## Hierarchie der Simple Types



## Simple Type Restrictions

- Simple Types können durch *Restriction* von anderen Simple Types abgeleitet werden
  - der *Base Type* ist immer ebenfalls ein Simple Type
  - Wurzel dieser Hierarchie ist der *anySimpleType*
- Restrictions enthalten *Facets*
  - Facets sind durch vorgegebene Elemente definiert
    - es gibt 12 Arten von Facets
  - jede Restriction enthält 0-n Facets
    - Facets können wiederholt werden
  - wichtiges Werkzeug zur exakten Typ-Definition

## Primitive Types Facets (I)

string	length, minLength, maxLength, pattern, enumeration, whiteSpace
boolean	pattern, whiteSpace
float	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
double	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
decimal	totalDigits, fractionDigits, pattern, whiteSpace, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
duration	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
dateTime	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
time	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
date	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive

## Primitive Types Facets (II)

gYearMonth	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
gYear	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
gMonthDay	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
gDay	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
gMonth	pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive
hexBinary	length, minLength, maxLength, pattern, enumeration, whiteSpace
base64Binary	length, minLength, maxLength, pattern, enumeration, whiteSpace
anyURI	length, minLength, maxLength, pattern, enumeration, whiteSpace
QName	length, minLength, maxLength, pattern, enumeration, whiteSpace
NOTATION	length, minLength, maxLength, pattern, enumeration, whiteSpace

## Patterns (Regular Expressions)

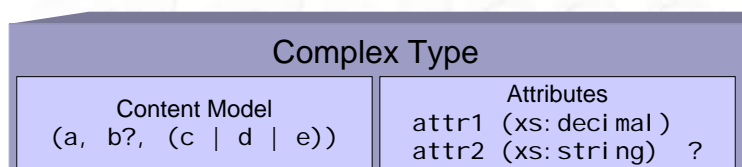
- Einschränkung von Simple Types
  - Restrictions mit dem pattern Element
- Beschränkung der lexikalischen Werte
- einfacher Aufbau der Ausdrücke
  - normale Zeichen: "C&A"
  - Zeichenklassen: "\p{I sBasi cLati n}"
  - Zeichenmengen: "[\p{I sBasi cLati n}-[\d]]"
  - Quantifiers: "[a-zA-Z]{1,8}"
  - Klammerausdrücke: "(XML(\s+|-))?Schema"
  - beliebige Kombination dieser Mechanismen

## Was ist ein Element?

- was ist ein Element in der Instanz?
  - was XML 1.0 definiert wie ein Element aussieht
  - zusätzlich dazu die Definition der XML Namespaces
- wie wird ein Element definiert?
  - ein Element hat (meist) einen *Complex Type*
    - es darf auch einen *Simple Type* haben
  - es kann *Identity Constraints* enthalten
  - es kann an drei Stellen definiert werden
    1. im Content Model eines *Complex Type*
    2. als eigenständiges Element zur Referenzierung
    3. innerhalb wiederverwendbarer *Named Model Groups*

## Was ist ein Complex Type?

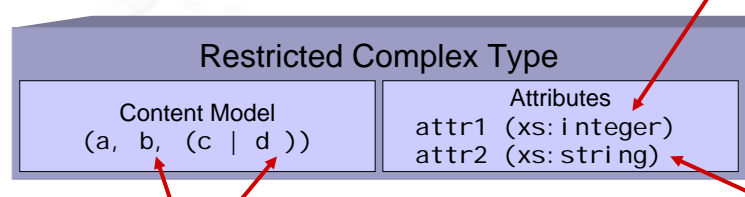
- definiert die Verwendung von Elementen
  - durch den erlaubten Inhalt (das *Content Model*)
  - und die erlaubten Attribute



- Complex Types können abgeleitet werden
  - *Restriction* schränkt Inhalt und/oder Attribute ein
  - *Extension* fügt Inhalt und/oder Attribute hinzu

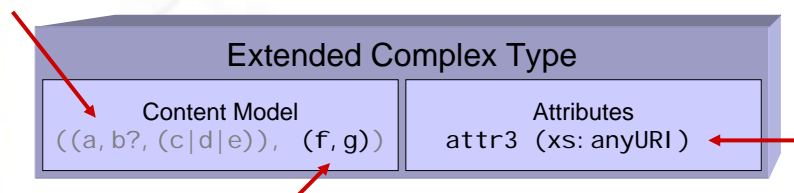
## Complex Type Restriction

- das Inhaltsmodell kann eingeschränkt werden
- die Attribute können eingeschränkt werden
  - ihre Verwendung oder ihre Typen
- Instanzen sind gültig gemäss des Basistyps
  - Code kann 1:1 wiederverwendet werden



## Complex Type Extension

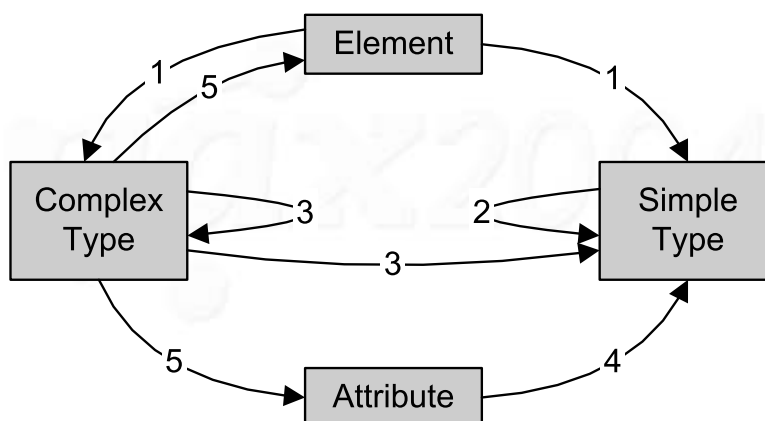
- Hinzufügen von Inhalt oder Attributen
  - Inhalt wird implizit an das Content Model angehängt
  - Attribute werden der Attributmenge hinzugefügt
- Instanzen sind ungültig gemäss dem Basistyp
  - Code muss mit der Erweiterung umgehen können



## Identity Constraints

- Erweiterung des ID/IDREF Konzepts
  - aber IDREFS wird nicht unterstützt
- bieten eine ganze Reihe von Vorteilen
  - belassen den *Typ* der Keys
    - ermöglicht eine Typ-Validierung der Keys
    - ermöglicht verschiedene Key Typen, z.B. Integers/Strings
  - beruhen auf *Wertgleichheit* ( $+002_{\text{integer}} = 2_{\text{integer}}$ )
  - ermöglichen Keys über *mehrere Felder*
  - ermöglichen *scoped* Keys (d.h. kontext-abhängig)
  - können auch auf *Elemente* angewendet werden
- sind etwas komplizierter zu benutzen

## XML Schema Components







## Lokale und Globale Deklarationen

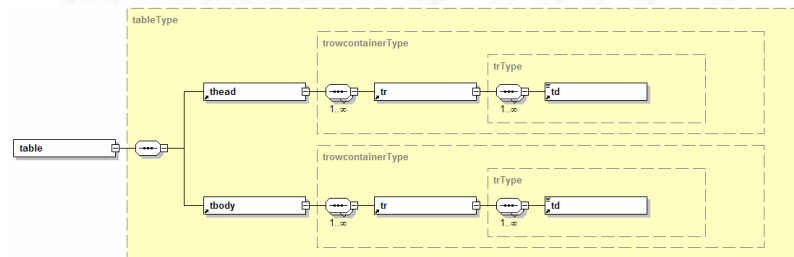
- XML Schema unterstützt Wiederverwendung
  - von Typen (simple und komplexe Typen)
  - von Elementen und Attributen
- eine Modellierungsfrage des Schemas
  - keine Auswirkungen auf die Instanzen
  - wichtig für langlebige Schemas
    - Wiederverwendung von Schemateilen
    - Versionierung
- beide Konzepte sind orthogonal
  - alle Kombinationen sind möglich und sinnvoll

## Orthogonalität der Konzepte

		Element/Attribut	
		<i>Global</i>	<i>Lokal</i>
Typen	<i>Global</i>	<b>1:</b> Element/Attribut und Typ sind benannt und wiederverwendbar	<b>2:</b> lokal definiertes Element/Attribut verwendet benannten Typ
	<i>Lokal</i>	<b>3:</b> Element/Attribut ist wiederverwendbar und verwendet einen lokal definierten Typen	<b>4:</b> lokal definiertes Element/Attribut verwendet lokal definierten Typen

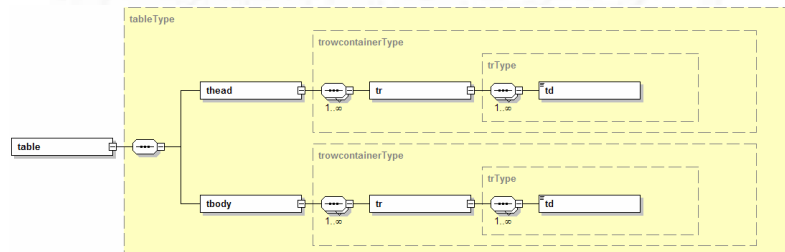
## 1: Alles Global

- globale Definition von Elementen und Typen
  - maximiert die Wiederverwendbarkeit
  - macht die XML Schema Source schwer lesbar
- mögliche Variante als generelles Vorgehen



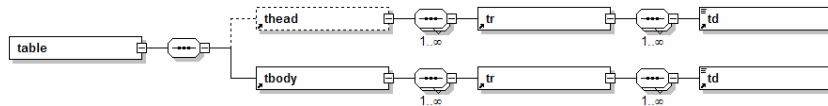
## 2: Globale Typen

- ermöglicht die Wiederverwendung von Typen
  - verschiedene Elemente mit gleichem Typ
  - ermöglicht Typen, die auf Typen basieren
- unterstützt strukturelle Gemeinsamkeiten



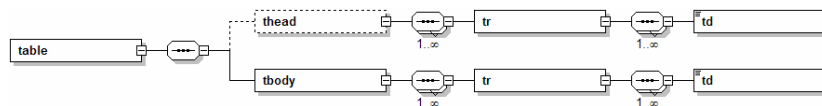
### 3: Globale Elemente/Attribute

- dies ist die einzig mögliche Variante in DTDs
  - aber nur fast, denn nur Elemente sind global
  - Attribute jedoch lokal (in der ATTLIST des Elements)
- gleiche Modellierungsprobleme wie in DTDs
  - Wiederverwendung von Elementen
  - die Struktur der Elemente ist nicht wiederverwendbar



### 4: Alles Lokal

- *Russian Doll* Modellierungsstil
  - alles ist ineinandergeschachtelt
    - maximiert Kompaktheit, reduziert Übersichtlichkeit
  - ermöglicht keinerlei Wiederverwendung
    - und damit auch keine Rekursion
- es existiert nur eine globale Deklaration
  - das Document Element ist global definiert



## Modellierungsfragen

- ist XML Schema meine Modellierungssprache?
  - falls ja, sollten Modellierungsregeln definiert werden
  - falls nein, wird es generiert aus dem Modell
- Modellierung für Wiederverwendung
  - Typbibliotheken für verschiedene XML Schemas
    - verschiedene Applikationen
    - verschiedene Versionen einer Applikation
  - Modellierungsrichtlinien sind wichtig
    - XML Schema bietet (zu) viele Freiheiten
    - Kontrolle z.T. über final, block und abstract
    - schlechte Modellierung kann sehr teuer werden

## Implementierungsfragen

- XML Schema benutzt Typen
  - viele Mechanismen beruhen auf der Typebene
  - Programme sollten eher Typen als Namen verwenden
- XML 1.0 kennt keine Typen
  - in sehr limitierter Weise und nur für Attribute
- Typbasierung beeinflusst durch Modellierung
  - *Russian Doll* Schema braucht keine Typen
  - globale Typen sollten erkannt werden
    - abgeleitete Typen mit abgeleitetem Code verarbeiten
  - XML Schema kennt sogar Casting (*Type Substitution*)

## XML Schema Support

- XML Schema Support in APIs
  - bisher kein etabliertes API für *PSVI Contributions*
  - bisher kein etabliertes API für *Schema Components*
- XML Schema Implementierungen
  - viele Parser implementieren XML Schema
  - kaum einer implementiert es komplett und korrekt
- XML Schema Test Suites vom W3C
  - <http://www.w3.org/2001/05/xml-schema-test-collection.html>
- IBM's *Schema Quality Checker (SQC)*
  - <http://www.alphaworks.ibm.com/tech/xmlsqc>

## Zusammenfassung

- XML Schema als Basis für XML
  - führt Typen ein
  - ermöglicht bessere (restriktivere) Schemas als DTDs
- XML Schema ist ein komplexer Standard
  - viele Modellierungsvarianten
  - Schemas werden schnell unübersichtlich
- XML Schema Modellierung ist wichtig
  - Basis für Datenbanken und verteilte Applikationen
  - bisher sehr wenig Arbeiten in diesem Bereich