Validierung als Pipeline

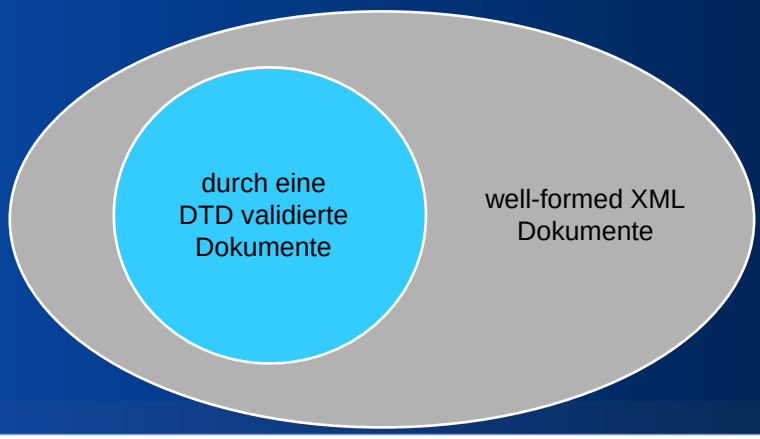
Kombination von XML Schemasprachen

Erik Wilde (ETH Zürich)

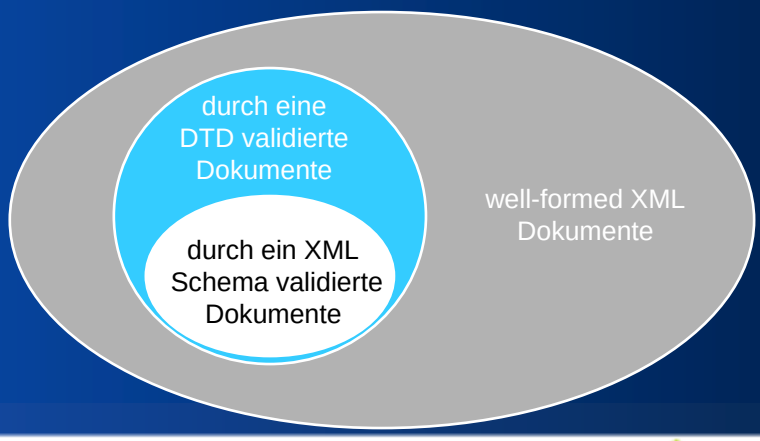
Gliederung

- XML Schemasprachen
  - Konzepte und Limitierungen
- Data Quality Assurance
- Validierung als mehrstufiger Prozess
  - Kombination verschiedener Schemas
- Erzeugung einer Validerungs-Pipeline
  - XML Processing Pipeline Languages
- Zusammenfassung

Schemasprachen als Konzept



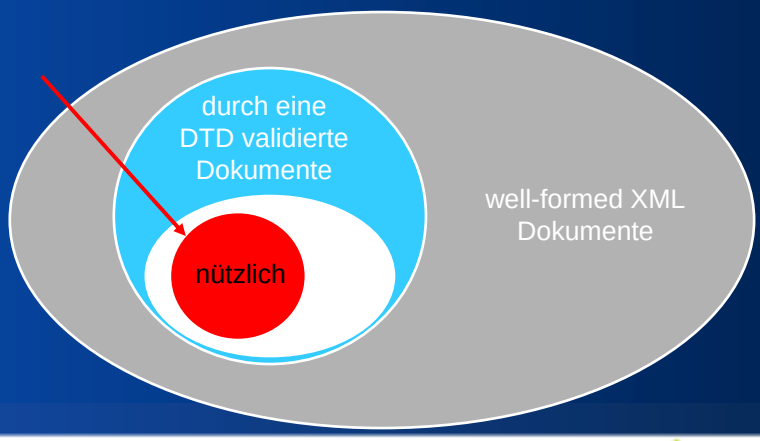
XML Schema als DTD++



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 Datenarbeiten
    - dass ein Attributwert in einer Datenbank existieren muss

Schema-valid vs. Application-valid



Data Quality Assurance

"Every application has a responsibility to ensure that only valid data is inserted into the repository. After all, what value would an application offer if the data it replied upon were corrupted?"

Chuck Cavaness (Programming Jakarta Struts)

Data Quality Handling

- Data Quality wird meist angenommen
  - "meine Daten sind immer korrektes XML"
- Data Quality in einem Workflow
  - meist stetig abnehmend
  - wird meistens hingenommen
- Data Quality Probleme kosten Geld
  - in der Entwicklung (Konzept und Infrastruktur)
  - und in der Anwendung (Processing Cycles)

Data Quality Firewalls

- entsprechend "normalen" Firewalls
  - existieren auf verschiedenen Netzwerkebenen
  - warum nicht auch auf Applikationsebene?
- an strategischen Stellen im Workflow
  - reduziert die Qualitätsverluste
- muss konfiguriert werden
  - Menge an eingebauten Validierungstools
  - u.U. Schnittstelle zu Business Logic Validierung

XML "Validity Levels"

- XML selber definiert zwei Levels
  - *well-formed* und *valid* XML Documents
- XML Schema definiert *schema validity*
- andere Standards funktionieren anders
  - RELAX NG processing ist nicht näher definiert
  - Schematron definiert Assertions und Reports
    - Teile von XSLT-Code
- Anwendungen brauchen bessere Konzepte

Mögliche XML "Validity Levels"

- not XML (syntax error, u.U. recoverable)
- well-formed, aber (external) entity errors
- well-formed
- character set validated
- character normalization validated
- valid (DTD)
- schema valid (partially oder fully)
- valid with additional rules (Schematron)
- business rule valid (DB lookups, ...)

Document Schema Definition Languages

- ISO Initiative für ein Schema-Framework
- definiert verschiedene Teile
  - grammar-based validation (RELAX NG)
  - rule-based validation (Schematron)
  - datatypes
  - path-based integrity constraints
  - character repertoire validation
  - declarative document manipulation (XSLT)

XML Character Validation

- viele Software hat Zeichensatz Limiten
  - Legacy Software ist häufig nur 8-bit fähig
- XML basiert auf Unicode
  - die spezifische Codierung ist nicht festgelegt
    - UTF-8 und UTF-16 müssen unterstützt werden
- XML kennt wenige Einschränkungen
- an anderen Stellen volle Freiheit
  - `#PCDATA` oder `xs:string` erlaubt alles
  - XML Schema und *mixed Content* problematisch

XML und Unicode

- XML erlaubt "beliebige" Unicode Zeichen

```
<文書 改訂日付="１９９９年３月１日">
  <題>サンプル</題>
  <段落>これはサンプル文書です。</段落>
  <!-- コメント -->
  <段落>&会社名;</段落>
  <図面 図面実体名="サンプル"/>
</文書>
```

CRVX (I)

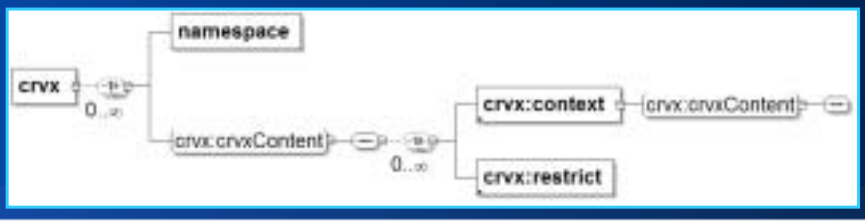
- Character Repertoire Validation for XML
  - einfache Schemasprache für Zeichensatz-Validierung
  - `http://dret.net/projects/crvx/`
- eine Variante für DSDL
  - bisher noch nicht offiziell präsentiert (WWW2003)
- Character Handling ist ein einfaches Problem
  - aber sehr häufig und wichtig zu beachten

CRVX (II)

- verschiedene Einschränkungen möglich
  - verschiedene syntaktische Strukturen
    - z.B. Element-Namen oder Attribut-Werte
    - berücksichtigt auch *mixed Content* (XML Schema nicht!)
  - verschiedene Kontexte eines Dokuments
    - verwendet XPath zur Identifikation von Kontexten
  - verschiedene Sichten auf XML
    - pure XML: keine Namespaces
    - Namespace-compliant XML: Prefixes und NCNames

CRVX Schema

- Schema für eine Schemasprache
  - definiert mit XML Schema
  - einfachster Weg für eine Definition
  - kann mit XML Tools verarbeitet werden



CRVX Implementierung

- CRVX Prototyp
  - basierend auf XSLT 2.0
    - XSLT 2.0 unterstützt *Regular Expressions*
  - kompiliert ein CRVX Schema in ein XSLT 2.0
  - dieses XSLT 2.0 ist der Validator
- Nachteile der Implementierung
  - XSLT 2.0 noch in Bewegung
- SAX wäre vermutlich eine bessere Basis

Pipelines als Verarbeitungsmodell

- XML ist ein Austauschformat
  - gemeinhin ist Austausch über das Netz gemeint
  - Austausch auf einem Rechner ist ein Spezialfall
- Pipelines aus der Unix Welt
  - Kopplung von vielen Tools
  - viele einfache Tools = viele Anwendungen
  - definierte Ausführungsumgebung
    - Ein- und Ausgaben und ein Fehlerkanal
    - Prozessmanagement

Unix Pipes

- Pipes werden in der Shell implementiert
  - Shell ist die Ausführungsumgebung
- Tools haben ein gemeinsames Datenmodell
  - Zeichenströme mit LF als Zeilentrennung
  - Probleme der Zeichencodierung (ASCII vs. UTF-8)
- die Shell übernimmt die Steuerung
  - Erzeugung der Prozesse (fork/exec)
  - Verbindung der Prozesse (stdin/stdout/stderr)

XML Pipelines

- XML ist ein zeichenbasiertes Format
  - das Datenmodell ist ein Zeichenstrom
- es gibt viele andere Informationsmodelle
  - DOM1, DOM2, DOM3
  - SAX1, SAX2
  - XML Information Set
  - XPath 1.0, XPath 2.0 (= XQuery 1.0)
- Pipelines verlieren häufig Informationen

XML 1.0 Pipelines

- Implementierungen
  - beliebige Tools, die XML 1.0 unterstützen
- Kopplung über viele Protokolle möglich
  - zeichenbasiert, also auch Unix Pipes
  - typische lose Kopplung von Komponenten
- extrem ineffizient
  - oft wiederholtes Parsen/Serialisieren
- Probleme mit out-of-band Signalen

SAX Pipelines

- SAX ist ein event-basiertes API
  - keine Datenstruktur, sondern Events
- je nach Aufbau ist Nebenläufigkeit möglich
  - effizientere Ausführung der Pipeline
  - auch SAX kann intern DOM verwenden
- existierende Implementierungen
  - Cocoon, das Apache XML Publishing Framework
  - Jelly, ein weiteres Apache-Projekt

DOM Pipelines

- DOM Bäume sind implementierungsspezifisch
  - DOM definiert nur ein Interface
  - das Speichermodell ist vollkommen offen
- viele Tools basieren auf DOM
  - egal ob XML Dokument oder "synthetisiertes" DOM
  - Pipeline benutzt DOM-Objekt (d.h. Speicherbereich)
- Implementierungen
  - DOM-Tools können kombiniert werden

Binary Infoset Pipelines

- nicht alle Tools benutzen das gleiche DOM
- nicht immer gibt es Shared Memory
- DOM Struktur effizient serialisieren
  - Persistent DOM (PDOM) Implementierungen
  - kein etabliertes Format
- sinnvoll bei speziellen Rahmenbedingungen
  - eigene Festlegung auf PDOM nötig
  - enge Bindung an Implementierung

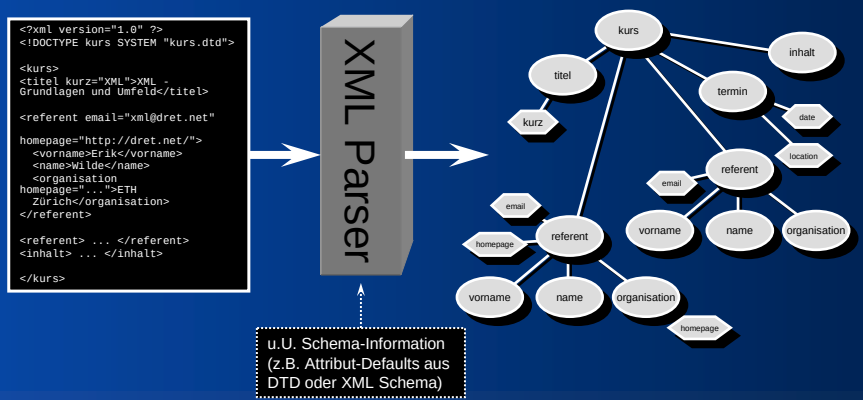
Heterogeneous Pipelines

- oftmals unterschiedliche Rahmenbedingungen
  - einige DOM-basierte Tools
    - effizient durch DOM-Objekte zu verbinden
  - Anbindung von Business Rules Validation
    - auf externem System implementiert
    - PDOM oder XML Dokument Anbindung
- flexibles Pipeline Framework
  - Komponenten deklarieren ihre Schnittstellen
  - Pipeline Definition basiert auf Kompatibilität

Validation Pipelines

- Spezialfall allgemeiner Processing Pipelines
  - u.U. stärker eingeschränkter Satz an Tools
    - Parser, XML Schema Processor, Schematron, CRVX, ...
- Business Rule Validation ist kompliziert
  - u.U. Anbindung an externe Datenbestände
  - oder allgemein Anbindung an externe Applikation
    - Pipeline Komponente mit einem Validation API
    - kann an beliebigen Code gebunden werden
    - aber nur ein Validation API benutzen (z.B. read-only)

XML Dokumente und Infoset/DOM



Sequence Matters

- Validation Pipelines kombinieren Schemas
  - sequentielle Abarbeitung der Pipeline
- die Reihenfolge kann wichtig sein
  - z.B. CRVX und Default-Werte aus einem Schema
- u.U. mehrfache Ausführung von Komponenten
  - Validation als Graph mit Ablaufsteuerung
    - Pipeline-Sprachen werden u.U. komplex
    - siehe *Web Services Flow Language (WSFL)*

Zusammenfassung

- Data Quality ist wichtig
  - oftmals vernachlässigt
  - Probleme verursachen langfristig hohe Kosten
- Validierung gehört zu einer Anwendung
  - auf verschiedenen logischen Ebenen
  - mit verschiedenen Tools
  - vereinigt in einer gemeinsamen Umgebung
- Entwicklung noch in der Frühphase

Q&A

Danke für Ihre Aufmerksamkeit!

Erik Wilde (ETH Zürich)