XML und RDF: Rückblick und Ausblick

Markus Stocker

25. Juni 2018

Rückblick

XML

- Extensible Markup Language
- Eine erweiterbare, beschreibende *meta* Auszeichnungssprache
- Hierarchisch (semi-) strukturierte Daten (Baumstruktur)

XML: Wichtige Sprachkonstrukte

- Tag
  - Beginnt mit < und endet mit >
  - Drei Arten: *opening*, *closing*, *empty element*
- Element
  - Beginnt mit *opening tag* und endet mit entsprechendem *closing tag*
  - Inhalt entweder Text oder andere Elemente
  - Beachte Gross-/Kleinschreibung, Sonderzeichen, Verschachtelung
- Attribut
  - In *opening* oder *empty-element tags*
  - Innerhalb Klammern < > als `name="value"` Paare
- Dokument
  - Enthält ein Wurzelelement
  - Ist wohlgeformt wenn syntaktisch korrekt

XML: Bemerkungen

- XML is mensch- und maschinenlesbar
- Tags sind allerdings nur für Menschen Bedeutungsvoll
- Verwendet für Datenaustausch, zwischen Anwendungen, im Internet
- Interoperabilität beruht allerdings auf Einigung (Schema)
- XML kann in Programmiersprachen gelesen/geschrieben werden
- XML hat auch Nachteile, insb. Ballast wegen tagging

XPath

- Ermöglicht die Verarbeitung von XML Daten
- Deklarativer Zugriff auf Teile eines XML Dokuments
- Selektion von Elementen und Inhalten
- Operationen auf Inhalten

XPath: Wichtige Sprachkonstrukte

- Lokalisierungspfad
  - Adressierung von Knotenmengen
  - Setzt sich aus mehreren Einzelschritten zusammen
  - Diese werden mittels / getrennt
  - Absolut/relativ und ausführlich/verkürzt
- Schritt
  - A::KT[P]*
  - Achse: Navigationsrichtung aus dem Kontextknoten
  - Knotentest: Gewünschte Knotenmenge
  - Prädikat: Filterbedingungen
- Prädikat
  - Filtrierung/Einschänkung der XPath Ergebnissmenge
  - Definition genauer Zielmenge
  - Komplexere Problemstellungen

DTD

- Sprache zur Spezifikation von XML Dokumente
- Beschreibung gültiger Elemente, Attribute, Struktur
- Diese werden in einer DTD deklariert
- Ermöglicht die Validierung von XML Dokumente
- Sprich diese auf Gültigkeit zu prüfen

DTD: Deklarationen

- Elemente
  - `<!ELEMENT name inhalt>`
  - Inhalt: Kindelemente, `PCDATA`, `EMPTY`
  - Inhaltsmodelle: Sequenz, Alternative, Wiederholungen
- Attribute
  - `<!ATTLIST element attribut typ wert>`
  - Typen: `CDATA`, enumerierte Liste, `ID`, `ENTITY`
  - Werte: Standardwert, `#REQUIRED`, `#IMPLIED`, `#FIXED value`
- Entitäten
  - `<!ENTITY name "wert">`

XML Schema

- Weitere Sprache zur Spezifikation von XML Dokumente
- Adressiert einige Schwächen der DTD
- Insb. Datentypen, Namensräume, XML Syntax
- Stellt vordefinierte Datentypen zur Verfügung
- Definition benutzerdefinierte Datentypen
- Ermöglicht detailliertere Spezifikation von XML Dokumente
- Strengere Restriktionen die auf Gültigkeit geprüft werden können

XML Schema: Datentypen

- Datentypen sind wichtig
- `2018-02-10` und `0.5` sind Daten von verschiedenem Typ
- In XML Schema spezifiziert man diese nicht als `PCDATA`
- Sondern als `xs:dateTime` und `xs:decimal`
- Zudem spezifiziert `xs:dateTime` das Format `YYYY-MM-DD`
- `2018-02-10` ist somit 10. Februar nicht 2. Oktober

XML Schema: Deklarationen

- Einfache Elemente (keine Kindelemente oder Attribute)
  - `<xs:element name="..." type="..."/>`
  - Typen: `xs:string`, `xs:boolean`, `xs:int`, ...
  - Vorgegebene (`default`) oder festgelegte (`fixed`) Werte
  - Häufigkeit des Elements (`minOccurs`, `maxOccurs`)
- Attribute
  - `<xs:attribute name="..." type="..."/>`
  - Können vorgegebene oder festgelegte Werte haben
- Komplexe Elemente
  - `<xs:element><xs:complexType>...`
  - Zwei Varianten, eine mit benutzerdefiniertem Datentyp
  - Ordnung mittels `xs:all`, `xs:choice`, `xs:sequence`

Wozu Schema

- Beispiel NASA, ESA und JAXA die Daten austauschen
- Einigung über verwendete Tags
- Wie auch die Struktur (Elemente, Attribute)
- Ziel ist es Unterschiede zu vermeiden
- Und so den Datenaustausch zu erleichtern
- Die Interoperabilität der Systeme erhöhen

Wohlgeformtheit und Gültigkeit

- Ein XML Dokument mit korrekter Syntax ist wohlgeformt
- Validiert ein wohlgeformtes Dokument einem Schema ist es gültig
- Ein gültiges Dokument ist immer auch wohlgeformt (*well-formed*)
- Ein wohlgeformtes Dokument ist nicht zwingend gültig (*valid*)

RDF

- Resource Description Framework
- Beschreibung von Ressourcen, digitale, physische, imaginäre Entitäten
- Grundlegender Baustein des *Semantic Web*
- Formale Spezifikation der Bedeutung von Daten
- Graphbasiertes Datenformat, Knoten und gerichtete Kanten
- Mittels URI benannt, Literale oder *blank nodes*

RDF: Literale und *blank nodes*

- Literale repräsentieren Datenwerte
- Können typisiert oder untypisiert sein
- Typisierte Literale haben einen Datentyp
- Untypisierte Literale können ein *language tag* haben
- *Blank nodes* sind unbenannte Ressourcen
- Haben strukturelle Funktion für mehrwertige Relationen

RDF: Tripel

- Die Elementareinheit in RDF
- Eine Struktur die aus drei Elementen besteht
- Das Subjekt, das Prädikat und das Objekt
- Subjekte und Objekte entsprechen Knoten
- Prädikate entsprechen gerichteten Kanten
- Prädikate sind immer benannt (URI)
- Subjekte und Objekte können unbenannt sein (*blank node*)
- Subjekte und Objekte können benannt sein (URI)
- Objekte können Literale sein

RDF: Syntax

- Tripelmengen können entsprechend einer Syntax serialisiert werden
- Besprochene Syntaxen: N-Triples, Turtle, RDF/XML
- Jede Syntax hat Vor- und Nachteile
- Man kann zwischen Syntaxen convertieren (automatisch)

SPARQL

- SPARQL Protocol And RDF Query Language
- Abfragesprache für RDF
- Ermöglicht deklarativer Zugriff auf RDF Daten
- *Tripel pattern* (Tripelmuster) fundamentale Struktur
- Besteht aus drei Elementen: Subjekt, Prädikat und Objekt
- Wobei diese auch variabel sein können (im Unterschied zum Tripel)

SPARQL

- Eine *triple pattern* Menge nennt man *basic graph pattern*
- Resultatformate: `SELECT`, `CONSTRUCT`, `ASK`, `DESCRIBE`
- Modifizierer: `FILTER`, `ORDER BY`, `LIMIT`, `OFFSET`
- SPARQL Update um RDF Daten deklarativ zu ändern
- SPARQL Endpoints für RDF Datenbank als Web Service
- Abfrageoptimierung damit man Antworten möglichst schnell erhält

RDF Schema

- RDFS unterstützt das Organisieren von Ressourcen einer Tripelmenge
- Grundlegend dabei ist die Gruppierung von Ressourcen
- Dafür stellt RDFS das Konstrukt der Klasse zur Verfügung
- RDFS ermöglicht Klassen als solche zu definieren
- Als Instanzen der Klasse aller Klassen (`rdfs:Class`)
- Dabei spielt das Prädikat `rdf:type` eine wichtige Rolle

RDF Schema

- RDFS ermöglich die Definition von Unterklassen (`rdfs:subClassOf`)
- Und somit die Erzeugung von Klassenhierarchien
- Prädikathierarchien sind ebenfalls möglich (`rdfs:subPropertyOf`)
- Klassenzugehörigkeiten (`rdfs:domain`, `rdfs:range`)

Ontologien

- Formalisierung von Wissen über eine Domäne
- Begriffe und Beziehungen eines Gegenstandsbereiches
- Informationsaustausch unter Beihaltung der Bedeutung
- Bestandteile: Klassen, Relationen, Instanzen
- Aufbau: Schema, Inhalt
- Wissensquellen: Mensch, Bücher, Web, Datenbanken

Ausblick

Übersicht

- XLink
- XPointer
- XSLT
- XQuery
- Web Ontology Language (OWL)
- Regeln
- Inferenz
- Programme
- Ontologien

XLink

- Ermöglicht das Setzen von Hyperlinks in XML Dokumente
- Ähnlich wie `<a href="">...</a>` in HTML
- Allerdings können beliebig benannte Elemente einen XLink setzen
- Von Browsern generell nicht unterstützt

```
<planets xmlns:xlink="http://www.w3.org/1999/xlink">
  <planet xlink:type="simple" xlink:href="http://planets.org">
    <name>Earth</name>
    <radius>6371</radius>
  </planet>
</planets>
```

XPointer

- Ermöglicht die Verlinkung auf Teile eines Dokuments
- XLink verlink nur auf ganze Dokumente

```
<!-- http://planets.org -->
<planets>
  <planet id="Earth">
    <name>Earth</name>
    <radius>6371</radius>
  </planet>
</planets>

<!-- My list of planets linking to http://planets.org elements -->
<planets xmlns:xlink="http://www.w3.org/1999/xlink">
  <planet xlink:type="simple" xlink:href="http://planets.org#Earth"/>
</planets>
```

XSLT

- eXtensible Stylesheet Language Transformations
- Styling Sprache für XML
- XML Daten für jegliche Anwendung zur Präsentation vorbereiten
- XML Dokumente nach andere XML Dokumente transformieren
- Aber auch nach (X)HTML oder andere Formate (z.B. CSV)
- XSLT verwendet XPath um XML Dokumente zu navigieren
- Elemente und Attribute können hinzugefügt/gelöscht werden
- Elemente können anders sortiert werden
- Man kann XML Dokumente client- oder serverseitig transformieren
- Clientseitig z.B. im Browser mittels JavaScript
- Serverseitig z.B. in PHP

XSLT Beispiel: Eingabe Dokument

```
<planets>
  <planet>
    <name>Earth</name>
    <radius>6371</radius>
  </planet>
  <planet>
    <name>Mars</name>
  </planet>
</planets>
```

XSLT Beispiel: Transformations Dokument

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

  <xsl:template match="/">
    <html>
      <body>
        <h2>Solar System Planets</h2>
        <table>
          <tr><th>Name</th><th>Radius</th></tr>
          <xsl:for-each select="planets/planet">
            <tr>
              <td><xsl:value-of select="name"/></td>
              <td><xsl:value-of select="radius"/></td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>

</xsl:stylesheet>
```

XSLT Beispiel: Ausgabe Dokument

```
<html>
  <body>
    <h2>Solar System Planets</h2>
    <table>
      <tr><th>Name</th><th>Radius</th></tr>
      <tr><td>Earth</td><td>6371</td></tr>
      <tr><td>Mars</td><td></td></tr>
    </table>
  </body>
</html>
```

XSLT Beispiel: Python

```
from lxml import etree

xslt = """ MY XSLT DOCUMENT """

xml = """ MY XML DOCUMENT """

transform = etree.XSLT(etree.XML(xslt))

print(str(transform(etree.XML(xml))))
```

XQuery

- Abfragesprache für XML
- Deklarativer Zugriff auf Elemente und Attribute
- Baut auf XPath auf und erinnert an SQL
- Kann verwendet werden um XML Daten zu transformieren

XQuery: Beispiel

```
for $x in doc("planets.xml")/planets/planet
where $x/radius>6000
order by $x/radius
return $x/name
```

Web Ontology Language (OWL)

- Eine weitere Ontologiesprache
- Baut auf RDF Schema auf
- Ist wesentlich ausdruckstärker
- Man kann damit also komplexeres Wissen formalisieren
- Unter Berücksichtigung der Rechenkomplexität
- Gleich wie RDFS sind auch OWL Ontologien RDF Dokumente
- Seit 2004 eine W3C Recommendation
- OWL 2 seit 2012 eine W3C Recommendation

Web Ontology Language (OWL)

- Unterklassen und ausserdem äquivalente und disjunkte Klassen
- Instanzen können als äquivalent oder verschieden deklariert werden
- Logische Klassenkonstruktoren ermöglichen komplexe Klassen
- Schnitt- (*and*), Vereinigungs- (*or*), Komplementärmenge (*not*)
- Prädikatbeschränkungen für komplexe Klassenbeschreibungen
- Äquivalente, inverse, transitive, symmetrische, funktionelle Prädikate

Regeln

- Wissen kann man in Ontologien auch als Regeln erfassen
- Schlussfolgerung wenn eine Prämisse erfüllt ist
- Beispiel: Eine Frau die ein Kind hat ist eine Mutter
- Regeln sind auch Teil einer Ontologie (des Schemas)
- Spielen auch eine wichtige Rolle bei Inferenz

Inferenz

- Bei der logischen Inferenz geht es um deduktive Schlussfolgerung
- RDFS, OWL und Regeln ermöglichen automatische Inferenz
- Einfaches Beispiel: Implizite Klasseninstanzen in einer Hierarchie
- Weiteres Beispiel: Konsistenzprüfung einer Ontologie
- Eines der komplexeren Themen im Bereich Wissensrepräsentation

Programme

- Protégé, TopBraid Composer, OntoStudio
- Stardog, Virtuoso, BlazeGraph, AllegroGraph
- HermiT, ELK, FaCT++
- OWL API, Jena, RDF4J
- ...

Ontologien

- Es gibt sehr viele existierende Ontologien
- Von abstraktem bis sehr konkretem Wissen
- Decken viele Themenbereiche
- Insbesondere Bioinformatik aber zunehmend auch andere
- Repositorien für Ontologien (z.B. EBI Ontology Lookup Service)
- Ontologien, Taxonomien, Thesauri
- Unterschiedlicher Komplexitäts- und Reifegrad