

XPath

Markus Stocker

19. März 2018

Rekapitulation

- Wie kommt es, dass ein XML Dokument eine Baumstruktur hat?
- Warum ist die Festlegung auf ein gemeinsames Vokabular wichtig?
- Was bedeutet Wohlgeformtheit?
- Wieviele Wurzelemente kann ein XML Dokument haben?
- Was ist
 - ▶ `<planet>`
 - ▶ `</planet>`
 - ▶ `<planet/>`
 - ▶ `name="Earth"`
 - ▶ `<planet>Earth</planet>`

Übersicht

- Was ist XPath
- XPath Konzepte
- Beispiele

Was ist XPath

- Mit XML kann man Daten strukturieren
- Man kann diese in mensch- und maschinenlesbarer Form speichern
- Nicht nur in Dateien sondern auch in Datenbanken
- Toll, aber letztlich muss man diese Daten flexibel verarbeiten können
- Dafür immer Programme zu schreiben ist mühsam
- Man benötigt Verarbeitungssprachen
- Davon gibt es mehrere, XPath ist eine

Was ist XPath

- Sprache zur Verarbeitung von XML Dokumenten
- Auf Teile eines XML Dokuments zugreifen
- Navigation durch Elemente und Attribute
- Selektion von Elementen und Inhalten
- Einfache Operationen auf Inhalten

Konzepte: Knoten (*node*)

- XPath modelliert ein XML Dokument als Knotenbaum
- Es gibt 7 Knotenarten
 - ▶ *Root node*, der virtuelle Elternknoten des Wurzelements
 - ▶ *Element node*, z.B. `<planet> ... </planet>`
 - ▶ *Attribute node*, z.B. `radius="6371"`
 - ▶ *Text node*, `CDATA`
 - ▶ *Namespace node*
 - ▶ *Comment node*
 - ▶ *Processing instruction node*
- Davon sind `element`, `attribute`, `text` die drei wichtigsten

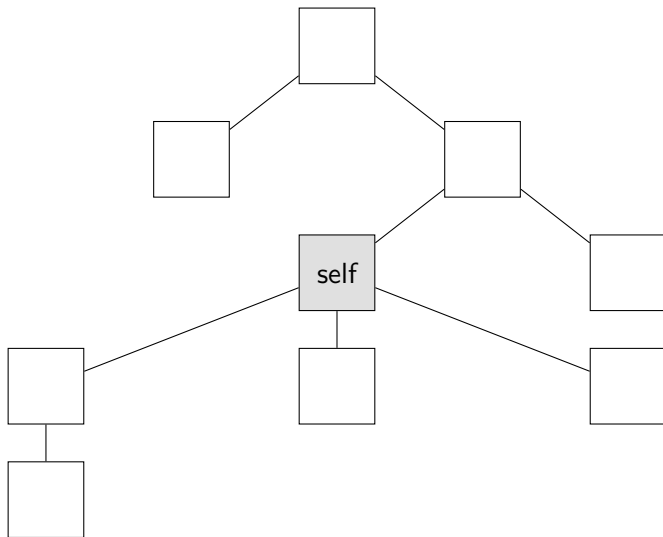
Konzepte: Achsen (*axis*)

- Navigation mittels XPath erfolgt aus einem Kontextknoten
- Achsen bezeichnen Verwandtschaftsverhältnisse zum Kontextknoten
- Dreizehn unterschiedlichen Achsen

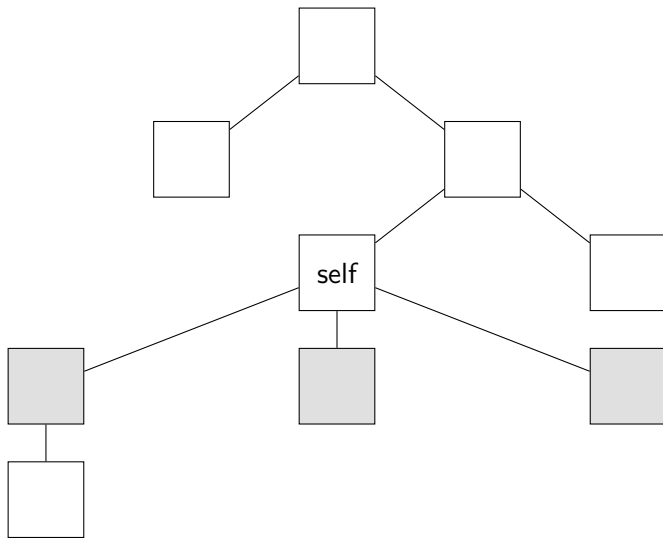
Achsen: Die Wichtigsten im Überblick

- *self*: Kontextknoten selbst
- *child*: Kindelemente des Kontextknotens
- *parent*: Elternelement des Kontextknotens
- *preceding-sibling*: Vorgängige Geschwister des Kontextknotens
- *following-sibling*: Nachfolgende Geschwister des Kontextknotens
- *ancestor*: Vorfahren des Kontextknotens
- *descendant*: Nachkommen des Kontextknotens

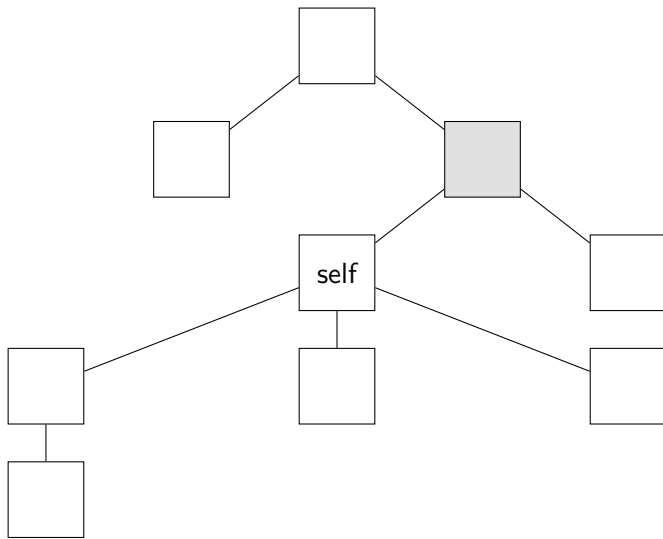
Achsen: *self*, der Kontextknoten



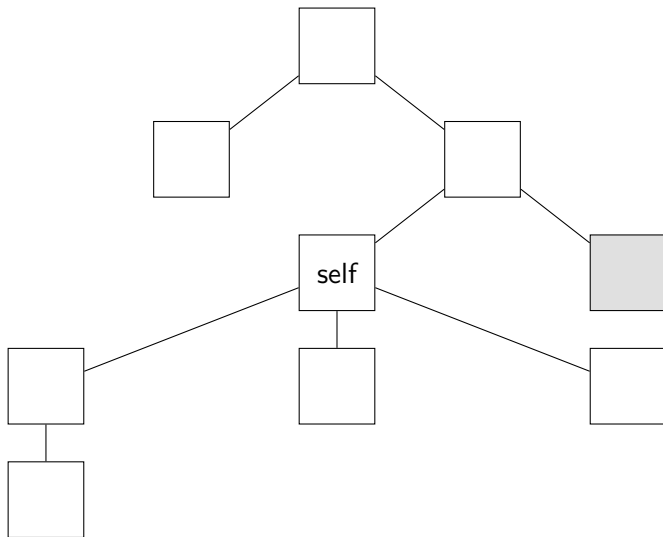
Achsen: *child*, die Kinder



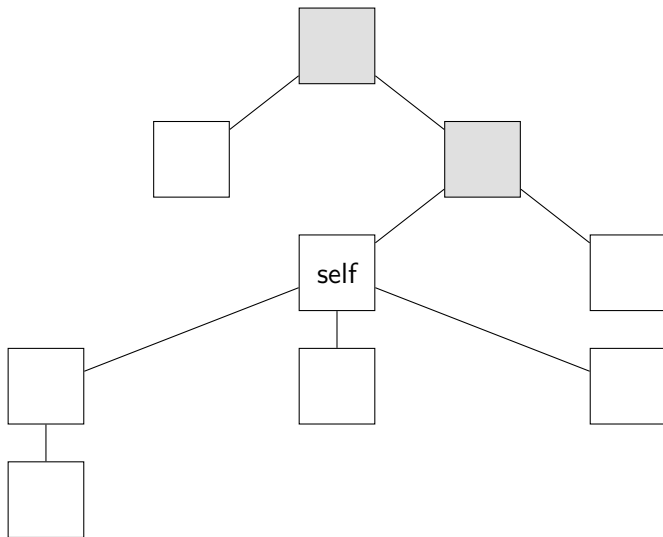
Achsen: *parent*, das Elternteil



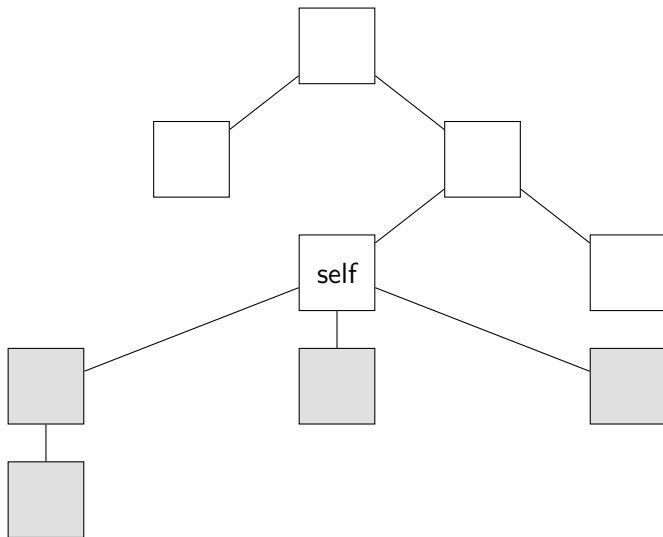
Achsen: *following-sibling*, die nachfolgende Geschwister



Achsen: *ancestor*, die Vorfahren



Achsen: *descendant*, die Nachkommen



Konzepte: Lokalisierungspfad

- Ermöglicht die Adressierung von Knoten (-mengen)
- Setzt sich generell aus mehreren Einzelschritten zusammen
- Diese werden durch Schrägstriche (/) voneinander getrennt

Lokalisierungspfad: Schritt (*location step*)

- Die Bestandteile eines Schrittes sind
- Achse: Richtung in die aus dem Kontextknoten navigiert wird
- Knotentest: Die gewünschte Knotenmenge
- Prädikat: Null, eins oder mehrere Filterbedingungen

`Achse::Knotentest[Prädikat]*`

Schritt: Beispiele

```
<planets>  
  <planet>  
    <name radius="6371">Earth</name>  
  </planet>  
</planets>
```

```
child::planets  
descendant::planet[name="Earth"]  
descendant::*
```

Lokalisierungspfade

- Kommen in ausführlicher oder verkürzter Form vor
- Können relativ oder absolut sein

Ausführliche und verkürzte Formen: Beispiele

Ausführliche Form	Verkürzte Form
<code>child::</code>	<i>Unterlassen</i>
<code>attribute::</code>	@
<code>descendant-or-self::node()/</code>	//
<code>self::node()</code>	.
<code>parent::node()</code>	..

Absolute Lokalisierungspfade

- Der erste Schrägstrich referenziert den Wurzelknoten

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

```
/child::planets/child::planet/child::name  
/planets/planet/name
```

Relative Lokalisierungspfade

- Der Pfad wird relativ zum Kontextknoten ausgewertet

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

```
child::planet/child::name  
planet/name
```

Adressierung verschiedener Knotentypen

<code>planet/name/text()</code>	Textknoten des Elements <code>name</code>
<code>planet/node()</code>	Alle Knotentypen (ausser Attribute)
<code>planet/comment()</code>	Kommentarknoten des Elements <code>planet</code>

Konzepte: Prädikate

- XPath Ergebnisse filtern
- Prädikate schränken die Ergebnisse ein
- Ermöglicht komplexere Problemstellungen
- Indem eine genauere Zielmenge definiert wird

Prädikate: Beispiel

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

```
/planets/planet/name[@radius]  
/planets/planet/name[@radius = "6371"]  
planet/name[@radius = 6371]  
planet/name[@radius="6371"]/text()  
planet[name]
```


Prädikate: Boolsche Operatoren

- Prädikate können boolsche Operatoren enthalten
- Insbesondere die Operatoren and und or

```
<planet>  
  <name radius="6371" temperature="14.9">Earth</name>  
</planet>
```

```
planet/name[@radius and @temperature]
```

```
planet/name[@radius=6371 and @temperature=14.9]
```

Prädikate: Boolsche Operatoren

```
<planets>
  <planet>
    <name radius="6371" temperature="14.9">Earth</name>
  </planet>
  <planet>
    <name radius="3389">Mars</name>
  </planet>
</planets>
```

planet/name[@radius=3389 and @temperature=14.9]

planet/name[@radius=3389 or @temperature=14.9]

Kaskadierende Prädikate

- Hintereinanderschaltung von Prädikaten
- Eine Knotenmenge wird gefiltert
- Es resultiert eine Zielmenge
- Diese ist Ausgangsmenge für das nächste Prädikat

```
<planet>  
  <name>Earth</name>  
  <radius>6371</radius>  
  <temperature>14.9</temperature>  
</planet>
```

```
/planet[name="Earth"] [radius=6371]
```

Vereinigungsmengen: Mehrere Knotentests

```
<planets>  
  <planet>  
    <name>Earth</name>  
    <radius>6371</radius>  
  </planet>  
  <planet>  
    <name radius="3389">Mars</name>  
  </planet>  
</planets>
```

```
planet[name="Earth"] | planet/name[@radius=3389]
```

Konzepte: Funktionen

- Erweiterte Operationen und Abfragen auf Knotentests
- Zum Beispiel, Stringanalyse von Textknoten und Attributwerten
- Verwendung in XPath-Ausdrücken oder Prädikaten
- Funktionen können ein Argument erhalten
- Geben immer einen Wert zurück

Funktionen: Beispiele

```
<planets>
  <planet>
    <name>Earth</name>
    <radius>6371</radius>
  </planet>
  <planet>
    <name radius="3389">Mars</name>
  </planet>
</planets>
```

```
planet[position()=2] (oder auch planet[2])
planet[starts-with(name, 'E')]/radius/text()
planet[not(radius)]/name/text()
count(planet)
planet[last()]
```

Zusammenfassung

- XPath unterstützt die Verarbeitung von XML
- Gezielt auf Inhalte zugreifen
- Adressierung mittels Lokalisierungspfade
- Resultierende Knotenmenge kann weiter eingeschränkt werden