XML Query Languages

Martin Steffen

IfI UiO

7. February 2007
Basics
  XML

Querying
  Intro
  Node selection queries & XPath
  Tree transformations & XSLT
  Iteration and joining: XQuery
  Querying by pattern matching: XDuce

Conclusion & perspective
Basics

XML

Querying

Intro

Node selection queries & XPath

Tree transformations & XSLT

Iteration and joining: XQuery

Querying by pattern matching: XDuce

Conclusion & perspective
XML

- extensible markup language

How to Buy a Wrench
There are two kinds of wrenches: wrenches with fixed size, and adjustable wrenches.
<html>
  <head>
    <title>Martin Steffen (Address)</title>
  </head>
  <body>
    <h4><a href="http://www.uio.no">University</a></h4>
    <p><table><tr><td><i>Department of Informatics</i></td></tr></table></p>
</body>
</html>
University

Department of Informatics
University of Oslo
P.O. Box 1080 Blindern
N-0316 Oslo, Norway

Visiting address:
Forskningsparken (i.e. the office is not in the main informatics building!)
Room 31154
History

- SGML: invented 1969, standardized 1980
- 1990: awareness for dynamic mark-up a la (S)GML at W3C
- 1996: first working draft of XML
- Feb. 1998: XML1.0 recommendation (later various versions)
- Feb. 2004: XML1.1
- 27. 1. 2007: official XQuery-semantics: [Draper et al., 2007]
XML-data = trees

```xml
<addrbook>
  <person>
    <name>Joe Doe</name>
    <email>doe@cba.org</email>
  </person>
  <person>
    <name>Jane Dow</name>
    <email>dow@egd.com</email>
    <tel>123-456-788</tel>
  </person>
</addrbook>
```
<addrbook>
  <person>
    <name>Joe Doe</name>
    <email>doe@cba.org</email>
  </person>
  <person>
    <name>Jane Dow</name>
    <email>dow@egd.com</email>
    <tel>123-456-788</tel>
  </person>
</addrbook>
<addrbook>
  <person>
    <name>Joe Doe</name>
    <email>doe@cba.org</email>
  </person>
  <person>
    <name>Jane Dow</name>
    <email>dow@egd.com</email>
    <tel>123-456-788</tel>
  </person>
</addrbook>
XML-data = trees

<addrbook>
  <person>
    <name>Joe Doe</name>
    <email>doe@cba.org</email>
  </person>
  <person>
    <name>Jane Dow</name>
    <email>dow@egd.com</email>
    <tel>123-456-788</tel>
  </person>
</addrbook>
XML-data = trees

```
addrbook
  ├── person
  │   ├── name
  │   └── email
  └── person
      ├── name
      └── email
tel
```
Trees with bells and whistles

- different data-models
- most important one: XPath data model
- complications:
  - different sorts of “nodes”:
    - element nodes (the “real” ones)
    - attribute nodes
    - text-nodes (= leaves)
    - comment nodes, processing instruction nodes, root node
- namespace mechanism [Bray et al., 2004a]
Trees with bells and whistles

<addrbook>
  <person name="Joe Doe">
    <email>doe@cba.org</email>
  </person>
  <person name="Jane Dow">
    <email>dow@egd.com</email>
    <tel>123-456-788</tel>
  </person>
</addrbook>
Trees with bells and whistles

addrbook

person
  name
    email
doe@cba.org

person
  name
    email
dow@egd.com
 tel
  123-456-789
Schema

- **schema**: specifying the allowed *form* of tree

```xml
<!ELEMENT addrbook person*>  
<!ELEMENT person (name, email*, tel?)>  
<!ELEMENT name #PCDATA>  
<!ELEMENT element #PCDATA>  
<!ELEMENT tel #PCDATA>
```
Schema

- **schema**: specifying the allowed *form* of tree
- **valid** XML-document: conforming to the given schema definition
- many different schema language (e.g. XMLSchema)

```xml
<!ELEMENT addrbook person*>  
<!ELEMENT person (name, email*, tel?)>  
<!ELEMENT name #PCDATA>  
<!ELEMENT element #PCDATA>  
<!ELEMENT tel #PCDATA>
```
Querying

Intro
Node selection queries & XPath
Tree transformations & XSLT
Iteration and joining: XQuery
Querying by pattern matching: XDuce

Conclusion & perspective
Design goals of XQuery

- **XQuery**: standard XML query language
- goals:
  - declarative
  - support of namespaces
  - “work together” with XMLSchema (support simple/complex datatypes)
  - support XML-syntax + human-readable syntax.
  - supports (static and dynamic) type checking

[Boag et al., 2003, Section 5, Conformance]

⇒ XQuery: designed to generalize SQL, as XML generalizes database tables
Relational tables & XML trees

people(firstname, lastname, age)

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Doe</td>
<td>34</td>
</tr>
<tr>
<td>Jane</td>
<td>Dow</td>
<td>19</td>
</tr>
<tr>
<td>Joe</td>
<td>Dew</td>
<td>33</td>
</tr>
<tr>
<td>Jim</td>
<td>Beam</td>
<td>44</td>
</tr>
</tbody>
</table>
Relational tables & XML trees

John Doe 34
Jane Dow 19
Joe Dew 33
Jim Beam 44
Relational tables & XML trees

unranked
From relations to trees

• obvious generalization of relational tables
  1. height two
  2. root with an unbounded number of child nodes
  3. nodes in the 2nd layer: fixed number of child nodes
Aspects of XML querying

querying = matching/extraction/navigation
       + iteration
       + recombination/ transformation
       + validation
Aspects of XML querying

validation: schema

navigation: XPath

iteration: XQuery

transformation: XLST
XPath [Clark and DeRose, 1999]

- typical **path-based** selection technology
- navigational primitives
- pinpoint where to capture data sub-components
- similar to path expressions in Lorel [Abiteboul et al., 1997] and Cω [Biermann et al., 2005]
- current version: XPath2.0
  - part of **XQuery** and **XSLT**
  - quite more expressive than XPath1.0
\[ \varphi = \text{path expression} = \text{predicate on paths} \]
XPath-navigation

- in principle: node selection by predicates on paths
- more concretely: sequence of location steps:

\[
\text{axis :: nodetest } [\text{expr}_1] [\text{expr}_2] \ldots
\]
Node tests and predicates

1. **axis**: probe into the tree along paths
2. **node test**: check for the nature of the node (text, element etc)
3. **predicates**: general expressions (checking attributes, values, comparisons, variables ...)

- evaluation
  - left to right
  - result(s): sequences of nodes.
Axes

- parent
- child
- descendant
- ancestor

- foll. sibling
- prec. sibling
- following
- preceding
XSLT

- XSL = *extensible stylesheet language*, XSLT = *XSL transformations*\(^1\)
- original purpose: XML $\mapsto$ HTML, i.e., “generalized stylesheets”
- current version XSLT2.0: full-fledged *query* language [Clark, 1999]
- contains XPath2.0

\(^1\)There is additionally XSL-FO for physical layout.
<xsl:template match="b:card">
  <html>
    <head>
      <title><xsl:value-of select="b:name/text()"/></title>
    </head>
    <body bgcolor="#ffffff">
      <table border="3">
        <tr>
          <td>
            <xsl:apply-templates select="b:name"/><br/>
            <xsl:apply-templates select="b:title"/><p/>
            <tt><xsl:apply-templates select="b:email"/></tt><br/>
            <xsl:if test="b:phone">
              Phone: <xsl:apply-templates select="b:phone"/><br/>
            </xsl:if>
          </td>
          <td>
            <xsl:if test="b:logo">
              <img src="business_card_files/uri.html"/>
            </xsl:if>
          </td>
        </tr>
      </table>
    </body>
  </html>
</xsl:template>

<xsl:template match="b:name|b:title|b:email|b:phone">
  <xsl:value-of select="text()"/>
</xsl:template>

<xsl:stylesheet>
<xsl:stylesheet version="2.0"
   xmlns:xsl="http://…"
>

<xsl:template match="…">
  ...
  <xsl:apply-templates select="…"/>
  ...
</xsl:template>

....

<xsl:stylesheet>
set of template rules

\[
\begin{align*}
\text{left-hand side:} & \quad \text{pattern match} \\
\text{right-hand side:} & \quad \text{sequence constructor } = \text{output} + \text{recursive descent} \\
\text{matching strategy:} & \\
1. & \quad \text{select all matching left-hand sides} \\
2. & \quad \text{select: most specific one} \\
3. & \quad \text{evaluate: sequence constructor (by recursion)} \\
\text{pattern:} & \quad \text{(restricted) XPath expression}
\end{align*}
\]
Complications

- **modes**: states of a tree transducer
- **iteration**: similar as in XQuery (see later)
- **variables** ⇒ parameter passing
- of lesser impact: names and direct calls of templates, priorities, copying nodes, grouping, functions etc.
Strings / trees

- String \((n \leq 1)\)
- Ranked tree \((n \leq 2)\)
- Unranked tree (*)

Ordered trees: sequence of children
From unranked to ranked

- many “classical” results on ranked trees
  [Rozenberg and Salomaa, 1996] [Thomas, 1990]
  [Comon et al., 2005]

⇒ two approaches
  1. generalize theories
  2. reduce to the unranked case
From unranked to ranked

unranked tree

1

2 3 4 5

6

7

9 10 11

8
From unranked to ranked

unranked tree

1

2 3 4 5

6

7

9 10 11
From unranked to ranked

unranked tree

1
2 3 4 5
6 7
9 10 11
From unranked to ranked

unranked tree ⇔ rank 2

note: there are other encodings possible, too
String automaton
String automaton

Diagram of a string automaton with transitions labeled 'a', 'b', and 'c'. The current state is marked with a red dot.
String automaton
String automaton
String automaton
String automaton
String automaton
String automaton
Automata on trees

```
  a
 / \
c   c
 / \
 a   a
 / \
|   |
|   |
|   a
|   |
|   |   b
```
Automata on trees
Automata on trees
Automata on trees
Automata on trees
Parallel vs. sequential machines
Parallel vs. sequential machines
Parallel vs. sequential machines
Parallel vs. sequential machines: tree walking
Transduction

```xml
<xsl:stylesheet ...>
  <xsl:template match="a", mode="q0">
    ...
  </xsl:template>
  <xsl:apply-templates select="..." mode="q1"/>
  ...
</xsl:template>

<xsl:stylesheet>
```
Transduction

<xsl:stylesheet ...>
  <xsl:template match= "a", mode = "q_0">
    <node> ... </node>
  </xsl:template>
  ... <xsl:apply-templates select= "..." mode="q_1"/>
</xsl:stylesheet>
Transducers

- machines so far: acceptors: (yes/no)
- needed: transducers

\[ a/b \]
Transducers

- machines so far: acceptors: (yes/no)
- needed: transducers
Transducers

- machines so far: acceptors: (yes/no)
- needed: transducers
Transducers

- machines so far: acceptors: (yes/no)
- needed: transducers

Diagram:

```
  a
 b
 a
 c
 a
```

```
  b
 a
```

```
 a/b
 a
```
Transducers

- machines so far: acceptors: (yes/no)
- needed: transducers
Transducers

- machines so far: acceptors: (yes/no)
- needed: transducers
Transducers

- machines so far: **acceptors**: (yes/no)
- needed: **transducers**
Generalization on trees

- analogous to the generalization from automata to trees.
- tree transducer:

\[ q(f(x_1, \ldots, x_n)) \rightarrow h(q_1(x_1), \ldots, q_n(x_n)) \]
Formal tree transducer models

- still no consensus
- many models
- **tree walking**
  - used to model XSLT [Bex et al., 2002], (and type checking [Milo et al., 2000]) [Neven, 2000]
  - [Bex et al., 2002] formal model based on tree walking transducer
  - expressive **pebble** automata/transducer, used for
    - XML-QL [Deutsch et al., 1999]
    - Lorel [Abiteboul et al., 1997]
    - UnQL [Buneman et al., 1996]
    - StruQL [Fernandez et al., ]
    - fragment of XSLT
  - still open: relationship to tree automata
    [Engelfriet et al., 1999] [Engelfriert and Hoogeboom, 1999] [Neven and Schwentick, 2000]
XQuery2.0

• current W3C proposal
• in a nutshell (i.e., besides practical aspects)
  extension of XPath2.0 by
  • constructing of
  • joining of
  • iterating over

XML-data
Expressions

- "standard" data expression
- XPath expressions (navigation)
- XML-constructing expressions
- **FLWOR**-expressions
  - iteration
  - join
FLWOR

- for: iteration
- let: binding
- where: filter
- order
- return: construction

• cf. SELECT from SQL, also for in XSLT

```xml
<floury>
  { for $r in fn:doc("recipes.xml")//
    rcp:recipe [./rcp:ingredient[@name="flour"]]
    return <dish>{$r/rcp:title/text()}
  }
</floury>
```
XSLT vs. XQuery

- different original intentions
  ⇒ different pragmatically strengths
- as full languages: both Turing complete [Kepser, 2004]

<table>
<thead>
<tr>
<th></th>
<th>XSLT</th>
<th>XQuery</th>
</tr>
</thead>
<tbody>
<tr>
<td>origin</td>
<td>style sheet language</td>
<td>SQL ...</td>
</tr>
<tr>
<td>status</td>
<td>W3C recommendation</td>
<td>W3C recommendation</td>
</tr>
<tr>
<td>abstraction level</td>
<td>lower level, procedural</td>
<td>declarative</td>
</tr>
<tr>
<td>strength:</td>
<td>XML-tree transformations by recursive</td>
<td>querying, joins</td>
</tr>
<tr>
<td></td>
<td>descent</td>
<td></td>
</tr>
<tr>
<td>human readable</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>syntax</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Querying, validation, & XML-processing

- schema validation (and static typing) optional in XQuery
- static vs. dynamic typing
- based on the analogy

```
schema  =  type
validation  =  type checking
```
Static typing and pattern matching for querying

- pioneered by XDuce [Hosoya and Pierce, 2003]
- two alternative ways of deconstructing XML-docs, from different “communities” [Castagna, 2005]
  - vertical: “path”-expressions (XPath most notably)
  - horizontal: regular expression pattern matching
- challenges
  - ML type system is not expressive/flexible enough for XML-data
  - adapt pattern matching + type inference
  - subtyping
  - advanced: parametric polymorphism
Pattern matching

```latex
let \((x, y) = (1, "abc")\) in \(e'\)
```

- **data value** `(1, "abc")`, pattern `(x, y)`,
- value
  - matched against pattern
  - result: substitution / binding: `x \mapsto 1, y \mapsto "abc"`
- pattern: value, some subterms replaced by (capture) variable
- match, using first match

```latex
match e with
  | (_,_,) \to true
  | _ \to false
```

- wildcard `_` ("don’t care")
**XDuce**

- first-order domain-specific functional language
  [Hosoya et al., 2002]
- influential language design
- foundation for static typing of XML-processing
- static type-checking
  - often only data/dynamic type checking
  - mainstream of XML-processing: only data checking, validating parser
  - no systematic connection: program ↔ type

⇒ regular expression type
Example

<addrbook>
  <person>
    <name>Joe Doe</name>
    <email>doe@cba.org</email>
  </person>
  <person>
    <name>Jane Dow</name>
    <email>dow@egd.com</email>
    <tel>123-456-788</tel>
  </person>
</addrbook>
type Addrbook   = addrbook[Person*]
type Person    = person[Name, Email*, Tel?]
type Name      = name[String]
type Email     = email[String]
type Tel       = tel[String]
Regular expression types

\[
T ::= () \quad \text{empty sequence}
\]

\[
X \quad \text{type variables}
\]

\[
l[T] \quad \text{label}
\]

\[
T, T \quad \text{concatenation}
\]

\[
T \mid T \quad \text{union}
\]

\[
\emptyset \quad \text{empty set/type}
\]

- **regular** expressions as derived forms: \( T^*, T?, T+ \).

- **well-formedness** restriction on the variables, ensuring regularity.
Acceptance relation

\[
\begin{align*}
&\frac{t : M(X)}{t : X} \quad \text{T-STATE} \quad \frac{}{(\,) : (\,)} \quad \text{T-EMPTY} \\
&\frac{v : T_1}{v : T_1 | T_2} \quad \text{T-OR}_1 \quad \frac{v : T_2}{v : T_1 | T_2} \quad \text{T-OR}_2 \\
&\frac{t_1 : X_1 \quad t_2 : X_2}{l[t_1, t_2] : l[X_1, X_2]} \quad \text{T-LAB}
\end{align*}
\]
Pattern matching and type inference

- good/usual approach: type checking = type inference + subtyping
- avoid excessive annotations
- local type inference, locally precise

assume input \texttt{Person} = \texttt{person[Name, Email*, Tel?]}
match expression:

```haskell
match p with
    \texttt{person[name[n],tel[t]]} 
    \rightarrow \ldots
| \texttt{person[name[n],rest]} 
    \rightarrow \ldots
```

inferred types:
- \(t, n\): strings
- \texttt{rest}: ?
Pattern matching and type inference

- good/usual approach: type checking = type inference + subtyping
- avoid excessive annotations
- local type inference, locally precise

assume input \texttt{Person} = \texttt{person[Name, Email*, Tel?]}
match expression:

```
match p with
  person[name[n],tel[t]]
  -> ...
| person[name[n],rest]
  -> ...
```

inferred types:
- \( t, n \): strings
- \textbf{rest}: (Email*, Tel?)
Pattern matching and type inference

- good/usual approach: type checking = type inference + subtyping
- avoid excessive annotations
- local type inference, locally precise

assume input \texttt{Person = person[Name, Email*, Tel?]}

match expression:

```
match p with
    person[name[n],tel[t]]
    -> ...
| person[name[n],rest]
    -> ...
```

inferred types:
- \( t, n \): strings
- \texttt{rest}: (Email+, Tel?) \| ()
Core idea

- pattern and types are **tree automata**
- patterns are types (or tree automata) with capture variables
- core algorithmic question: given input type $T$ and pattern $P$

\[
\Pi \vdash T \triangleright P_1 \mid P_2:: \Pi_2; (\Gamma_1 \mid \Gamma_2)
\]

- note: intersection and difference of tree automata (resp. tree languages)
- take care: recursion
Core idea

- pattern and types are **tree automata**
  - patterns are types (or tree automata) with capture variables
  - core algorithmic question: given input type $T$ and pattern $P$
    - do they **match**? if so: **infer** the most-precise types for the matching vars.

\[
\Pi \vdash (T \cap P_1) \triangleright P_1 : \Pi_1 ; \Gamma_1 \quad \Pi_1 \vdash (T \setminus P_1) \triangleright P_2 : \Pi_2 ; \Gamma_2
\]

\[
\Pi \vdash T \triangleright P_1 \mid P_2 : \Pi_2 ; (\Gamma_1 \mid \Gamma_2)
\]

- note: intersection and difference of tree automata (resp. tree languages)
- take care: recursion
Type inference

\[
\begin{align*}
\Pi \vdash (T \cap P_1) \triangleright P_1 &:: \Pi_1; \Gamma_1 & \quad & \Pi_1 \vdash (T \setminus P_1) \triangleright P_2 &:: \Pi_2; \Gamma_2 \\
\Pi \vdash T \triangleright P_1 \mid P_2 &:: \Pi_2; (\Gamma_1 \mid \Gamma_2) \\
\Pi \vdash T_1 \triangleright P &:: \Pi_1; \Gamma_1 & \quad & \Pi_1 \vdash T_1 \triangleright P &:: \Pi_2; \Gamma_2 \\
\Pi \vdash T_1 \mid T_2 \triangleright P &:: \Pi_2; (\Gamma_1 \mid \Gamma_2) \\
I(X_1, X_2) \not\subseteq \emptyset \\
\Pi \vdash X_1 \triangleright Y_2 &:: \Pi_1; \Gamma_1 & \quad & \Pi_1 \vdash X_2 \triangleright Y_2 &:: \Pi_2; \Gamma_2 \\
\Pi \vdash I(X_1, X_2) \triangleright I(Y_1, Y_2) &:: \Pi_2; (\Gamma_1 \mid \Gamma_2)
\end{align*}
\]
Further properties

- type checking = type inference + subtyping
- semantic subtyping
  - type = set of values/trees
  - subtyping = inclusion of tree languages
- more advanced features:
  - parametric polymorphism = generic schemas
  - basis for pattern-matching querying optimization
- adopted widely: CDuce/CQL, Xtatic, Scala, XHaskell
- also: basis of type checking for XQuery
Basics
XML

Querying
Intro
Node selection queries & XPath
Tree transformations & XSLT
Iteration and joining: XQuery
Querying by pattern matching: XDuce

Conclusion & perspective
Trends

- robust theory
- algebra
- domain specific languages
- optimization
- non-exact matching
Left out

- XML-technologies:
  - name spaces
  - most of actual schema languages: XMLSchema, DTD (but there are many more)
  - parts of the technology covered has been simplified, especially XSLT
Material I

Primary sources for XML and related technologies is the web, especially the site of the W3C, where the recommendations and proposals are available. A general up-to-date book about XML is [Møller and Schwartzbach, 2006]. The standard references for XQuery are [Boag et al., 2003] [Frankhauser et al., 2001]. A formal description of XPath and XQuery is also available as official W3C recommendation since recently [Draper et al., 2007]. XSLT is described in [Clark, 1999]. Concerning typing and pattern matching . . . , the original material is [Hosoya and Pierce, 2003] [Hosoya et al., 2002] [Hosoya et al., 2005], and also the thesis [Hosoya, 2001]. Background literature on tree automata and related issues can be found in [Rozenberg and Salomaa, 1996] [Comon et al., 2005] [Thomas, 1990]. Specifically in the context of XML, I consulted [Vianu, 2001], [Neven, 2002b] (or a bit deeper [Neven, 2002a]). A short comparison of various language designs for various XML query languages from various research groups present in the W3C XQuery working group (and thus influential on the design of XQuery) can be found [Fernandez et al., 1999].
References

   The Lorel query language for semistructured data.

   A formal model for an expressive fragment of XSLT.

   The essence of data access in Cω.
   In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages (ECOOP)*.

   W3C Recommendation.

   *XQuery 1.0: an XML Query Language*.
   W3C Working draft.

   *Namespaces in XML 1.1*.
   W3C Recommendation, available at www.w3c.org/TR/xml11.

   *Extensible Markup Language (XML) 1.1*.
   W3C Recommendation.

   A query language and optimization for unstructured data.
   In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, volume 25 of
Patterns and types for querying XML documents.  
Keynote talk DBPL - XSYM 2005.

XSLT Transformations (XSLT).  
W3C.  
W3C Recommendation.

XML path language (XPath).  
http://www.w3c.org/TR/xpath.  
W3C Recommendation.

Tree automata, techniques and application.  
Available electronically.

XML-QL: A query language for XML.  
In WWW The Query Language Workshop (QL).

XQuery 1.0 and XPath 2.0 formal semantics.  
available at http://www.w3.org/TR/xquery-semantics/.  

Tree-walking pebble automata.  
In Jewels are Forever. Contributions to Theoretical Computer Science in Honor of Arto Salomaa, pages 72–83.  
Springer-Verlag.
Trips on trees.
*Acta Cybernetica*, 14:51–64.

Catching a boat with a strudel: Experiences with a web-site management system.

XML query languages: Experiences and exemplars.
Technical report, Bell Labs.

Xquery 1.0. formal semantics.
www.w3.org/TR/query-semantics.

*Regular Expression Types for XML*.
PhD thesis, University of Tokyo.

Parametric polymorphism for XML.
In *Proceedings of POPL ’05*, pages 55–62. ACM.

XDuce: A typed XML-processing language.
References IV

Regular expression types for XML.
*ACM Transactions on Programming Languages and Systems*.

A simple proof of the Turing-completeness of XSLT and XQuery.
In *Extreme Markup Languages*.

*XML Schema Requirements*.
W3C Note.

Typechecking for XML transformers.

*An Introduction to XML and Web Technologies*.
Addison-Wesley.

On the power of walking for querying tree-structured data.

Automata, logic, and XML.

Automata theory for XML researchers.
*SIGMOD Record*, 31(3).
References V

   On the power of tree-walking automata.

   Handbook of Formal Languages.
   Springer.

   Automata on infinite objects.
   Handbook of Theoretical Computer Science, Volume B.

   W3C Recommendation.

   From Codd to XML.
   In ACM PODS, Santa Barbara, CA.