Graduiertenkolleg Konstanz
Summer School Gaschurn, September 26-29 2006

Workshop
Processing and visual exploration of XML data in BaseX

Christian Grün, Alexander Holupirek, Marc Kramis

Department of Computer and Information Science
University of Konstanz

Overview

motivation

topic of the workshop

encoding

how xml is internally stored

visualizing

xml in fullscreen

file system

encoding file systems in xml

visual idefix

analyze query access patterns

Motivation

We'll present…

…how we store and access XML data

…how our XML encoding can be applied to build visualizations

…how files and file systems can be represented in XML

…how query access patterns can be analyzed in Idefix

We'll discuss…

…whether our ideas can be applied to your domains

…your ways of processing tree structures

…which visualizations you choose for hierarchic data structures

Motivation

XML – heard before?

- XML documents are flat representations of textual tree structures
- trees: simple, ordered, acyclic graphs with one root node
- existing node types: element nodes (tags) & text nodes
- XML attributes “belong” to element nodes

Encoding XML Documents

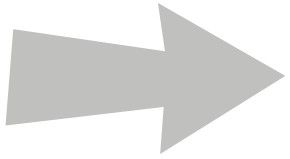
- two (major) alternatives:
  - rebuild tree structure via objects: DOM (Document Object Model)
  - map XML nodes to tuples (flat, »relational« encoding)

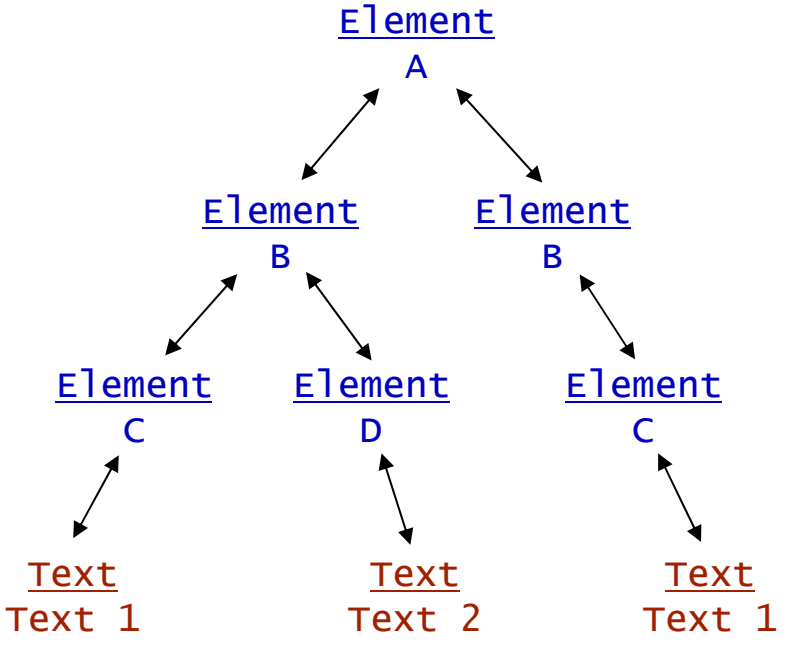
  → based on XPath Accelerator concept by Torsten Grust

Storage

DOM Encoding

```
<A>
    <B>
        <C>Text 1</C>
        <D>Text 2</D>
    </B>
    <B>
        <C>Text 1</C>
    </B>
</A>
```





Characteristics

+ comfortable access to all XML nodes

– all nodes have variable sizes (e.g. different number of children)

– redundant information (parent ↔ children)

Storage

Tuple Based Encoding

- sequential storage of node properties
  e.g.: `Node ID, Node Type, String, Parent ID, Post, Size, Level`

```
<A>
   <B>
      <C>Text 1</C>
      <D>Text 2</D>
   </B>
   <B>
      <C>Text 1</C>
   </B>
</A>
```

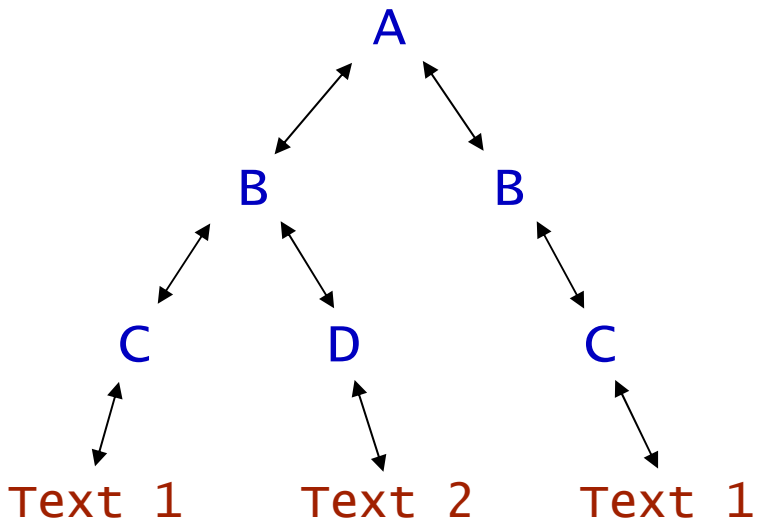
ID	Typ	String	Par	Post	Size	Lev
1	tag	A	–	9	9	1
2	tag	B	1	5	5	2
3	tag	C	2	2	2	3
4	txt	Text 1	3	1	1	4
5	tag	D	2	4	2	3
6	txt	Text 2	5	3	1	4
7	tag	B	1	8	3	2
8	tag	C	7	7	2	3
9	txt	Text 1	8	6	1	4

+ constant memory consumption, simple data structure

Storage

Tuple Based Encoding

Parsing the document: `<A><B><C>Text 1</C><D>Text 2...`



ID	Typ	String	Par	Post	Size	Lev
1	tag	A	–	9	9	1
2	tag	B	1	5	5	2
3	tag	C	2	2	2	3
4	txt	Text 1	3	1	1	4
5	tag	D	2	4	2	3
6	txt	Text 2	5	3	1	4
7	tag	B	1	8	3	2
8	tag	C	7	7	2	3
9	txt	Text 1	8	6	1	4

Storage

Tuple Based Encoding

- still variable size for strings

→ put tags & text contents in indexes and use numeric references

- still redundant data (e.g.: `Post = ID + Size - Level`)

→ limit storage to essential tree information

Selected Encodings

- `ID/Post/Parent` original XPath Accelerator concept (`Pre/Post`)
- `ID/Size/Level` used in MonetDB/XQuery and Idefix
- `ID/Parent` used in BaseX

Storage

Tuple Based Encoding: BaseX

- storage of `Type/String/Parent` suffices to query and reconstruct tree!

ID	Typ	String	Par	Post	Size	Lev
1	tag	A	–	9	8	1
2	tag	B	1	5	4	2
3	tag	C	2	2	1	3
4	txt	Text 1	3	1	0	4
5	tag	D	2	4	1	3
6	txt	Text 2	5	3	0	4
7	tag	B	1	8	2	2
8	tag	C	7	7	1	3
9	txt	Text 1	8	6	0	4

ID	Typ	Str	Par
1	0	1	–
2	0	2	1
3	0	3	2
4	1	1	3
5	0	4	2
6	1	2	5
7	0	2	1
8	0	3	7
9	1	1	8

Tags	
1	A
2	B
3	C
4	D

Texts	
1	Text 1
2	Text 2

- `ID` serves as table pointer → no explicit storage
- save more memory: each row is merged into 64 bits

Querying

```
<A>
    <B>
        <C>Text 1</C>
        <D>Text 2</D>
    </B>
    <B>
        <C>Text 1</C>
    </B>
</A>
```

ID	Typ	Str	Par
1	0	1	–
2	0	2	1
3	0	3	2
4	1	1	3
5	0	4	2
6	1	2	5
7	0	2	1
8	0	3	7
9	1	1	8

Tags	
1	A
2	B
3	C
4	D

Texts	
1	Text 1
2	Text 2

Example Query: `//C/text()`

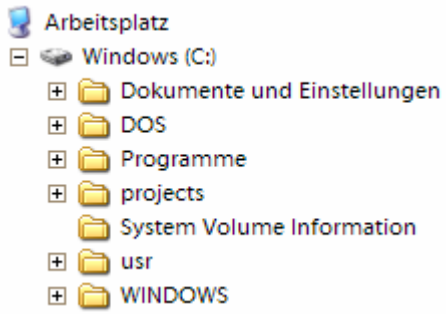
→ sequential scanning of node table

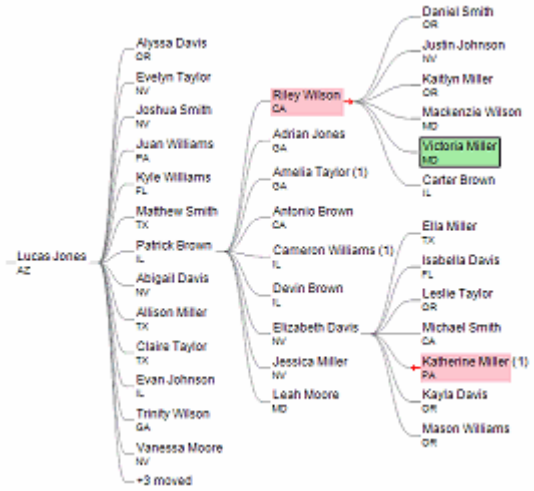
→ `//C` scanned nodes: `1-9` → `Str = 3` ? → `ID 3,8`

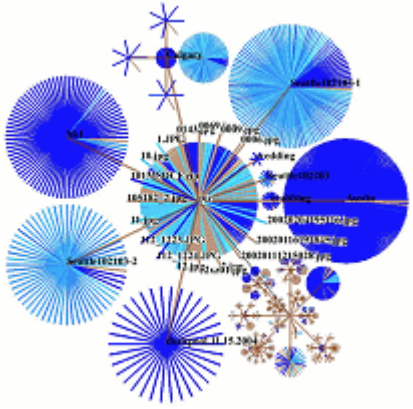
→ `text()` scanned nodes: `4,9` → `Typ = 1 & Par = 3/8` ? → `ID 4,9`

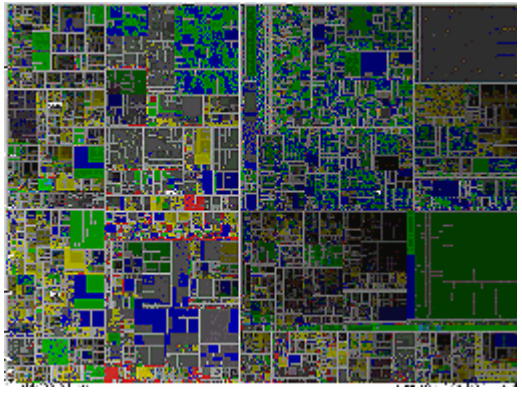
Visualization

```
<A>
   <B>
      <C>Text 1</C>
      <D>Text 2</D>
   </B>
   <B>
      <C>Text 1</C>
   </B>
</A>
```









Visualization

Hierarchic Views

- motivation: build hierarchic visualizations to show efficiency of encoding
- secondary (here): finding best visualization concept for XML data
- note: algorithms for querying and visualizing data are very similar
- most frequent operation: requesting children and parents of nodes

Creating Views

a) use available node table for all operations (drawing, interacting, etc.)
   → no additional data structures, saves memory

b) create supportive data structure for specific visualization
   → speeds up subsequent operations

Visualization

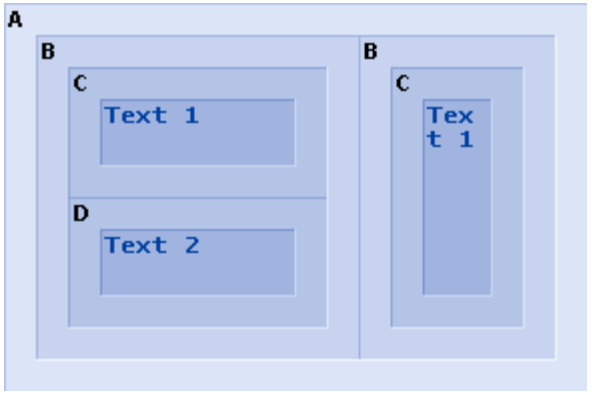
Example: TreeMap in BaseX

ID	Typ	Str	Par
1	0	1	–
2	0	2	1
3	0	3	2
4	1	1	3
5	0	4	2
6	1	2	5
7	0	2	1
8	0	3	7
9	1	1	8

Tags	
1	A
2	B
3	C
4	D

Texts	
1	Text 1
2	Text 2

ID	XS	YS	XE	YE
1	0	0	20	13
2	1	1	19	12
3	2	2	11	11
4	3	3	10	5
5	2	6	11	10
6	3	7	10	9
7	12	1	18	12
8	13	2	17	11
9	14	3	16	10



1) scan node table and create treemap rectangles
2) create visualization by drawing the rectangles and tags/text nodes
3) realize interactions by parsing the rectangles (focusing/selecting)
4) create new rectangles/new visualization for a new node set (zooming/filtering)

XML Data

XML can be used for many scenarios

- Popular XML Databases:
  DBLP (300 MB), Wikipedia (5 GB)
- Library Data, U Konstanz
  → MedioVis, HCI
- IP Network Data
  → TreeMap Visualizations, DBVis (Florian Mansmann)
- Mail Storage
  → Idefix/Mailix, DISY (Marc Kramis, Tim Petrowsky)
- File Handling
  → FSXML, DISY/DBIS…