# A Comparison of Element-based and Path-based Approaches to Indexing XML Data

**Michal Krátký**[1], michal.kratky@vsb.cz
**Radim Bača**[1], radim.baca@vsb.cz

[1]Dpt. of Computer Science, VŠB–Technical University of Ostrava

DATESO 2006

# Contents

- Introduction – XML, query languages, indexing XML data.
- XPath Accelerator (XPA).
- Multi-dimensional approach to indexing XML data.
- Experimental results.

# Introduction

- “Database view“: XML - approach to data modelling.
- Set of documents is a database, ***DTD*** (***XML Schema)*** is its database schema.
- XML query languages (***XPath***, ***XQL***, ***XQuery***,…).
- Common approaches to indexing XML data:
  - Element-based methods,
  - Path-based methods,
  - Sequence-based methods.

# XPath query

- XPath query consists of sequence of location steps
- Format of location step – axis::name[filter]
  - axis is a relation between nodes
  - context node

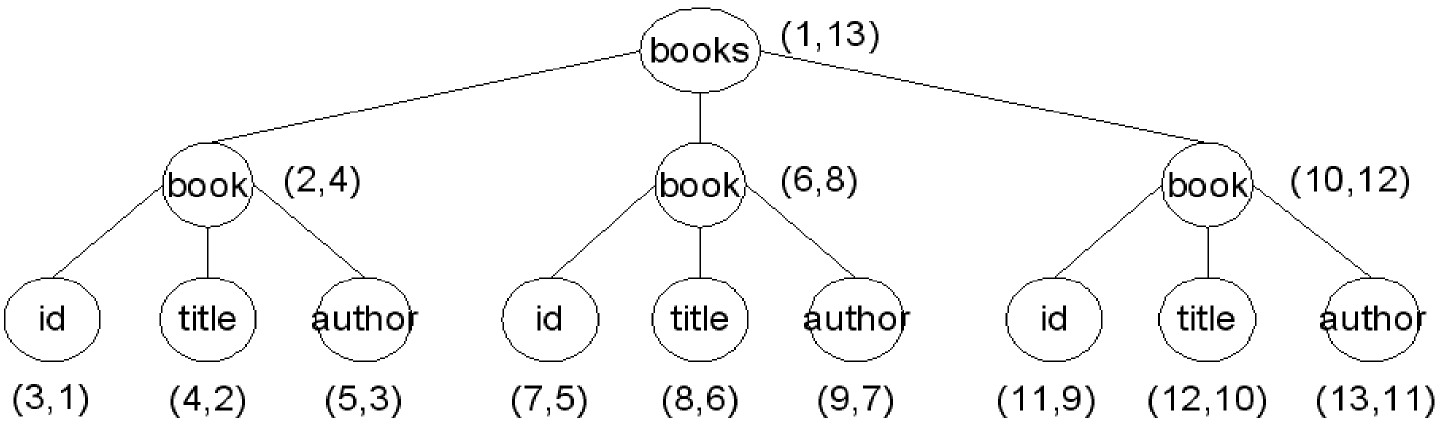
```
<?xml version="1.0" ?>
<books>
  <book id="003-04312">
    <title>The Two Towers</title>
    <author>J.R.R. Tolkien</author>
  </book>
  <book id="001-00863">
    <title>The Return of the King</title>
    <author>J.R.R. Tolkien</author>
  </book>
  <book id="045-00012">
    <title>Catch 22</title>
    <author>Joseph Heller</author>
  </book>
</books>
```

- Example:
  - /descendant::book[author = 'J.R.R. Tolkien']/child:title context node
  - //book[author = 'J.R.R Tolkien']/title

# XPA - model

- Dietz numbering scheme (preorder, postorder).
- Every XML node is valuated by a tuple:

  **(preorder, postorder, parent_preorder, attribute, $id_T$)**
- We can resolve axis relation for one node with single range query.



# XPA - indexes

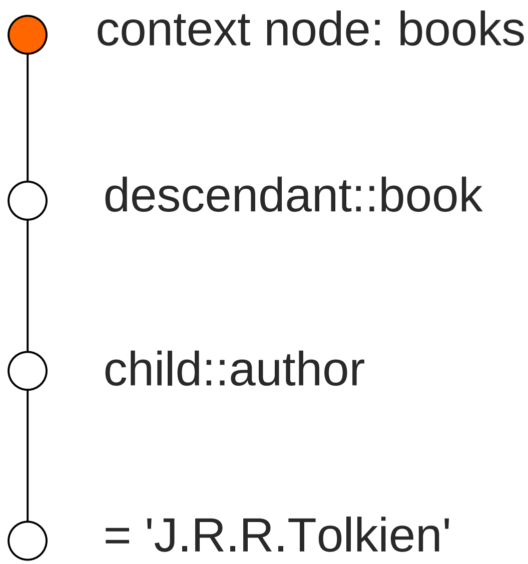
- ***Term index –*** map tags onto numbers.
- ***XPA index –*** a storage of all tuples of an XML document.
  - Multidimensional structures can be utilized for better performance.
- ***XML Content index –*** a storage of element and attribute content. We utilized inverted list.

# XPA - XPath query evaluation

- The query evaluation follows these steps:
  - axis::name resolution of the location step for every context node.
  - Result is a set of nodes. We resolve a filter if there is any.
  - The rest is considered to be context nodes for the next location step.
- The query is processed step by step.
- Result of the last location step is the result of the query

# XPA – query evaluation example

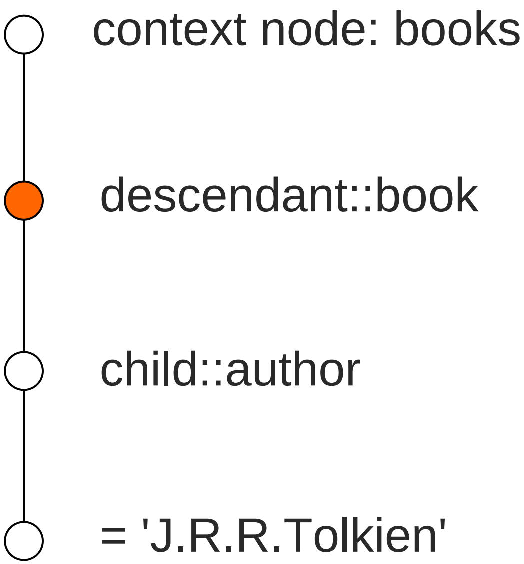
- **books/descendant::book[author = 'J.R.R. Tolkien']**




- We retrieve context nodes from **XPA index** with range query:
  `(0,0,0,0,0):(max,max,max,0,0)`

# XPA – query evaluation example

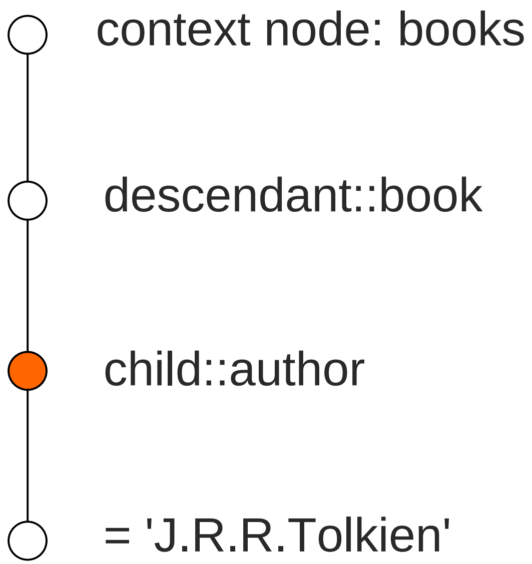
- **books/descendant::book[author = 'J.R.R. Tolkien']**




- Result: (1,13,_,0,0)
- Evalution of the next location step is done by range query:
  `(1,0,0,0,1):(max,13,max,0,1)`

# XPA – query evaluation example

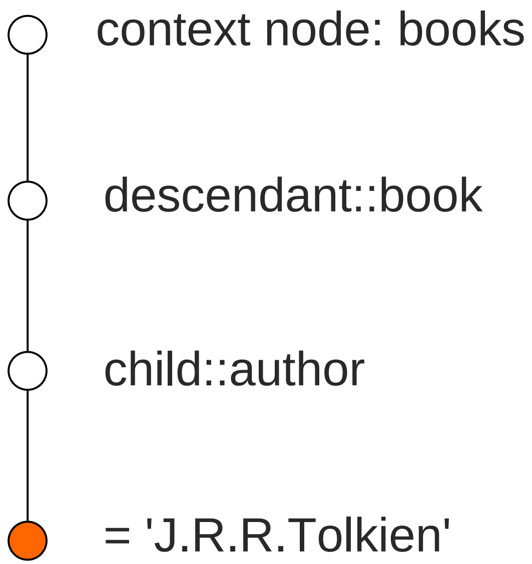
- **books/descendant::book[author = 'J.R.R. Tolkien']**




- Result: (1,13,_,0,0)
- Result: (2,4,1,0,1)
  (6,8,1,0,1)
  (10,12,0,1)
- Evalution of the next location step has to be done for each node from previous result:
  `(2,0,2,0,6):(max,4,2,0,6)`
  `(6,0,6,0,6):(max,8,6,0,6)`
  `(10,0,10,0,6):(max,12,10,0,6)`

# XPA – query evaluation example

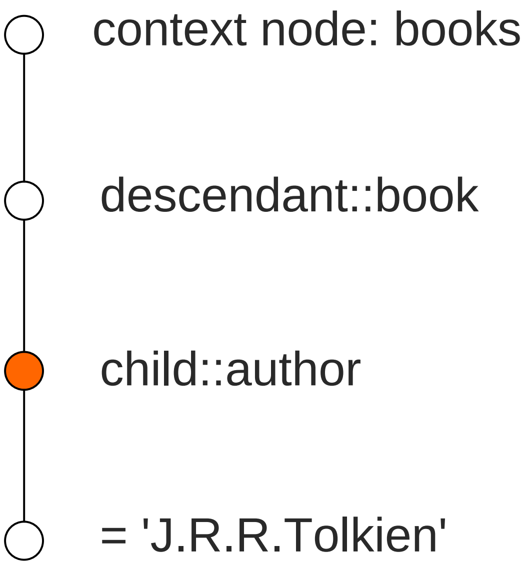
- **books/descendant::book[author = 'J.R.R. Tolkien']**




- Result: (1,13,_,0,0)
- Result: (2,4,1,0,1)
  (6,8,1,0,1)
  (10,12,0,1)
- Result: (5,3,2,0,6)
  (9,7,6,0,6)
  (13,11,10,0,6)
- From the **XML content index** we get preorder numbers of all elements with 'J.R.R.Tolkien' content.

# XPA – query evaluation example

- **books/descendant::book[author = 'J.R.R. Tolkien']**




- Result: (1,13,_,0,0)
- Result: (2,4,1,0,1)
  (6,8,1,0,1)
  (10,12,0,1)
- Result: (5,3,2,0,6)
  (9,7,6,0,6)
  (13,11,10,0,6)
- Result: {5,9}

# XPA – query evaluation example

- **books/descendant::book[author = 'J.R.R. Tolkien']**

context node: books

descendant::book

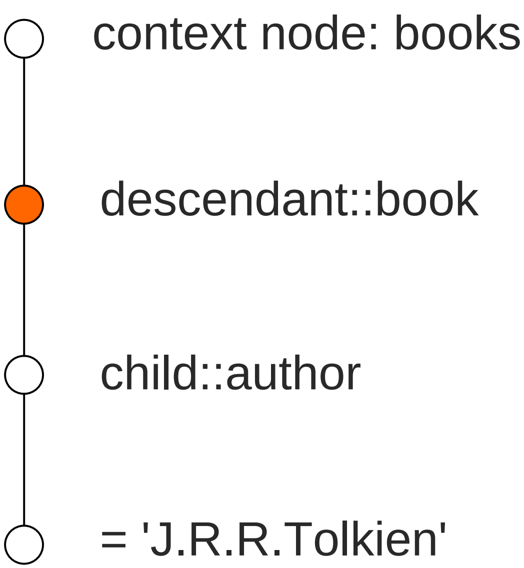
child::author

= 'J.R.R.Tolkien'

- Result: (1,13,_,0,0)
- Result: (2,4,1,0,1)
  (6,8,1,0,1)
  (10,12,0,1)
- Result: (5,3,2,0,6)
  (9,7,6,0,6)
  (13,11,10,0,6)
- Result: {5,9}

# XPA – query evaluation example

- **books/descendant::book[author = 'J.R.R. Tolkien']**

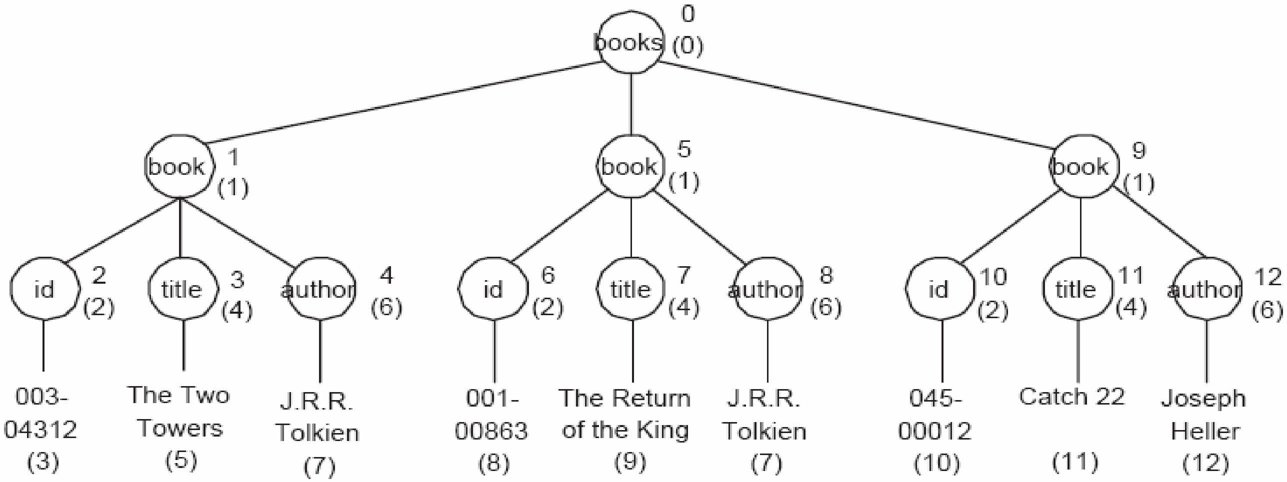



- Result: (1,13,_,0,0)
- Result: (2,4,1,0,1)
  (6,8,1,0,1)
  (10,12,0,1)
- Result: (5,3,2,0,6)
  (9,7,6,0,6)
  (13,11,10,0,6)
- Result: {5,9}

# Multi-dimensional approach to indexing XML data

- A graph is a set of the paths. XML document is decomposed to paths and labelled paths.
- ***labelled path***: $lp$: $s_0,s_1,\ldots,s_{lPN}$
- ***path***: $p$: $id_U(u_0),id_U(u_1),\ldots,id_U(u_{lLP}),s$

$id_U(u_i)$ – unique number of a node $u_i$

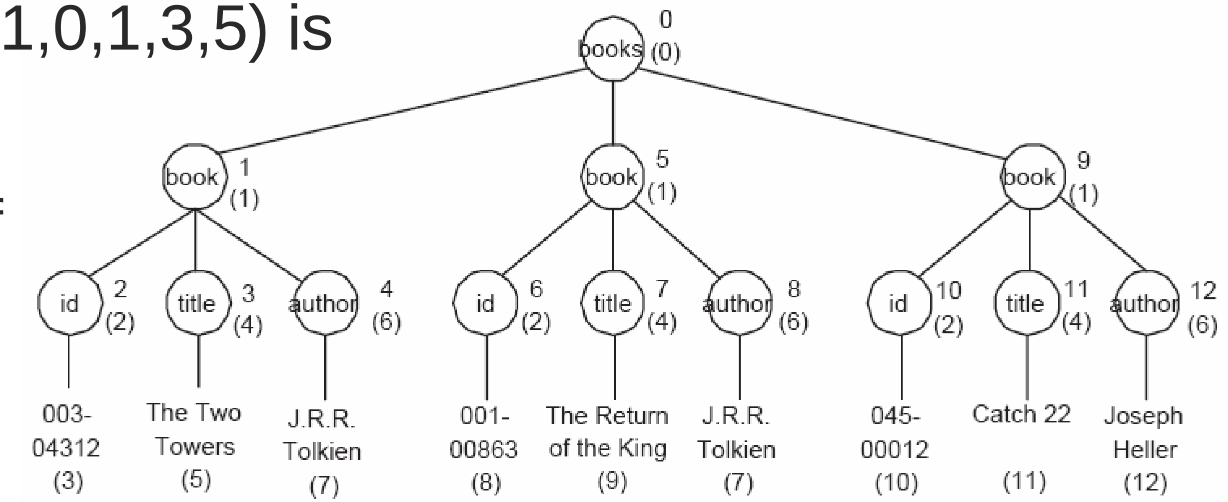


# Indexes

- ***Term index*** – a storage of strings $s_i$ of an XML document and their $id_T(s_i)$.
- ***Labelled path index*** – a storage of points representing labelled paths.
- ***Path index*** – a storage of points representing paths.

# Example
# labelled path index, path index

- Labelled paths:
  - `books,book,id` – point (0,1,2), $id_{LP} = 0$
  - `books,book,title` – point (0,1,4), $id_{LP} = 1$
  - `books,book,author` – point (0,1,6), $id_{LP} = 2$
- For example, the path to value `The Two Towers` belongs to the labelled path `books,book,title` with $id_{LP}$ 1. Vector (1,0,1,3,5) is created using $id_{LP}$, unique numbers $id_U$ of elements, and $id_T$ of the term.

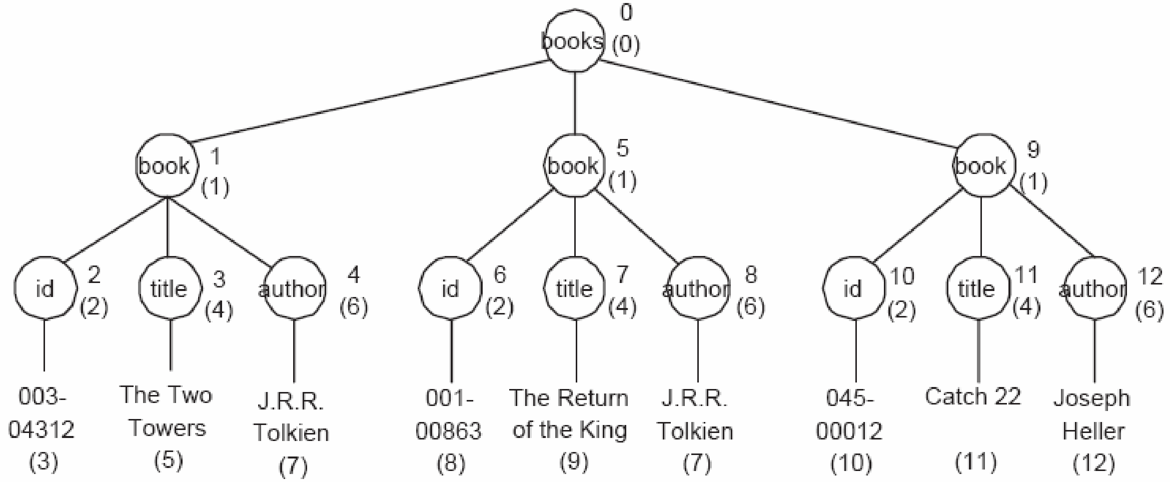


# Query for values of elements and attributes

- **books/book[author=“Joseph Heller”]**

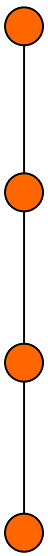
- context node: books (0)
- descendant::book (1)
- child::author (6)
- = ‘Joseph Heller’ (12)

- Search for appropriate labelled paths.



# Query for values of elements and attributes

- **books/book[author="Joseph Heller"]**



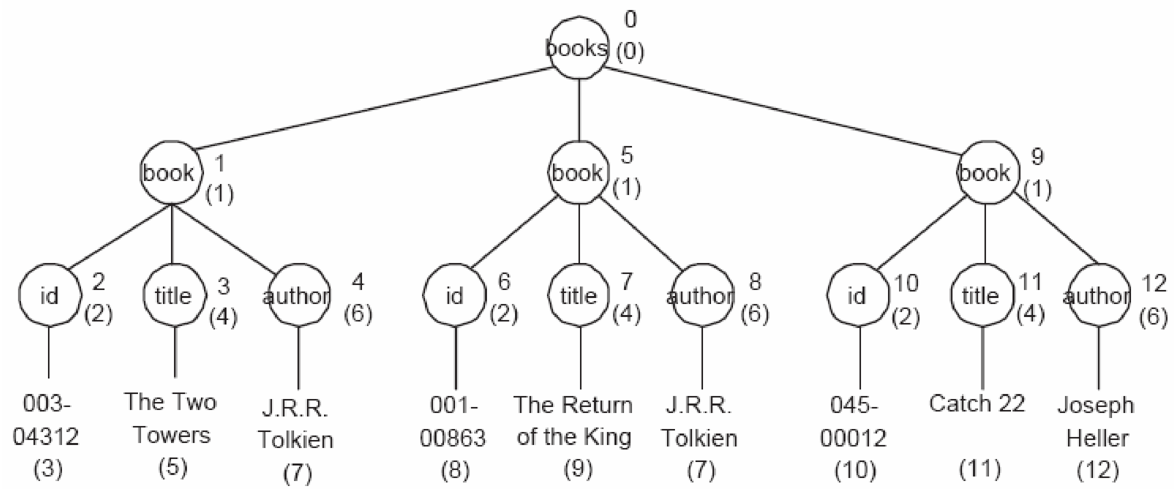
context node: books (0)

descendant::book (1)

child::author (6)

= 'Joseph Heller' (12)

- Result: `books,book,author (0,1,6)`
- Search in the ***labelled path index***: point query `(0,1,6)`



# Query for values of elements and attributes

- **books/book[author="Joseph Heller"]**

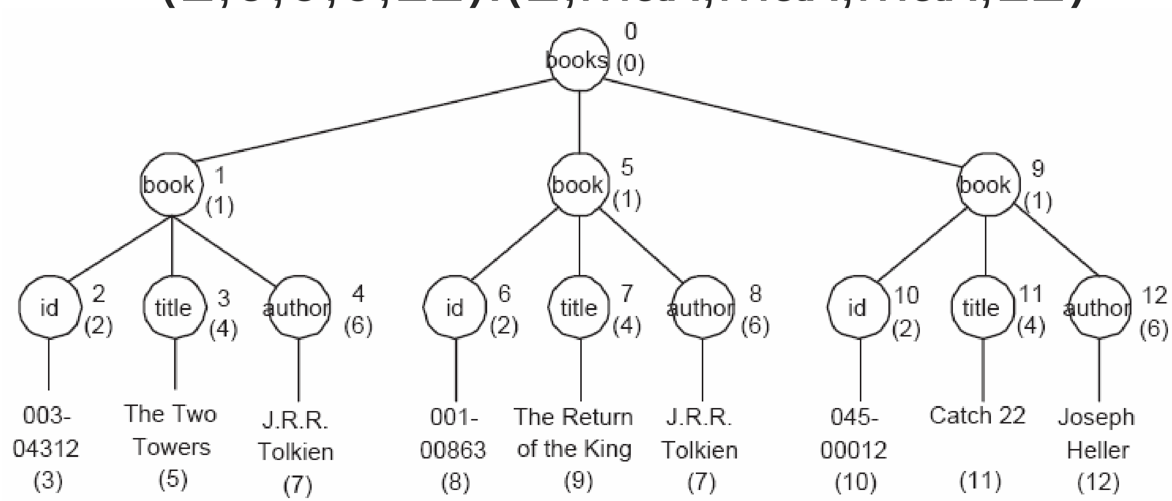
context node: books (0)

descendant::book (1)

child::author (6)

= 'Joseph Heller' (12)

- Result: `books,book,author (0,1,6)`
- Result: $id_{LP}$ 2
- Search points in the ***path index***: (2,0,0,0,12):(2,max,max,max,12)

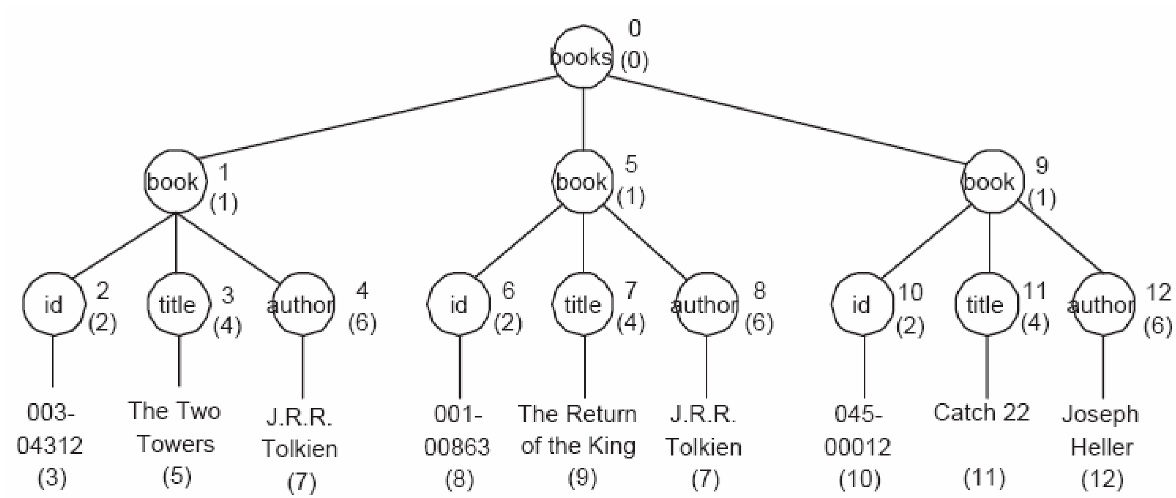


# Query for values of elements and attributes

- **books/book[author="Joseph Heller"]**

- context node: books (0)
- descendant::book (1)
- child::author (6)
- = 'Joseph Heller' (12)

- Result: `books,book,author (0,1,6)`
- Result: $id_{LP}$ 2
- Result: $id_{U}$ 0,9,12



# Query for values of elements and attributes

- **books/book[author="Joseph Heller"]**

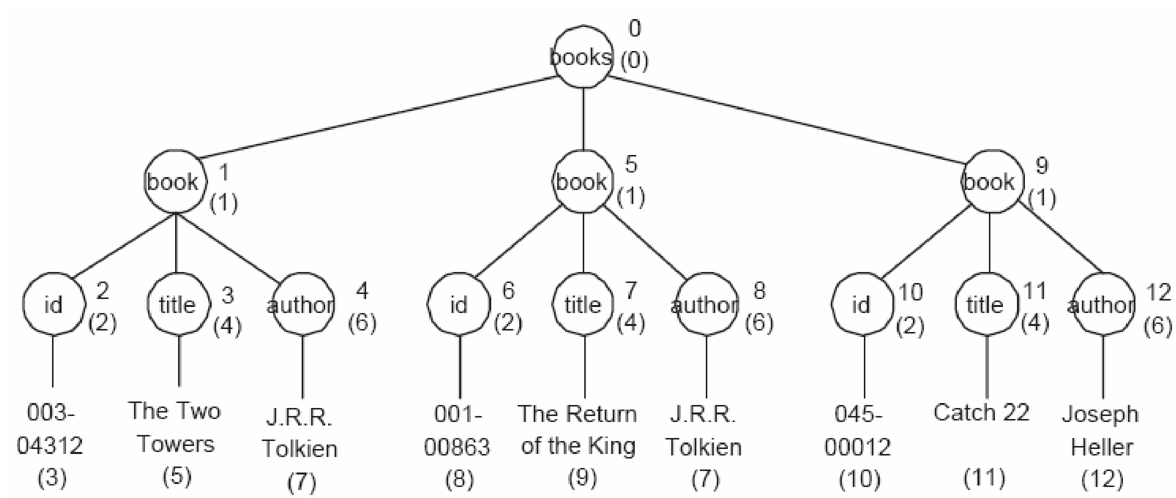
context node: books (0)

descendant::book (1)

child::author (6)

= 'Joseph Heller' (12)

- Result: `books,book,author (0,1,6)`
- Result: $id_{LP}$ 2
- Result: $id_U$ 0,9,12 – desired element



# Index data structures

- Paged and balanced multi-dimensional data structures – ***(B)UB-trees***, variants of ***R-trees***.
- They provide point and range queries.
- Problems:
  - *narrow range query* – the signature is applied for efficient processing – Signature R-tree.
  - indexing points with *different dimensions* – ***BUB-forest, R-forest.*** Each tree indexes space of different dimension***.***

# Experimental results

- XMARK collection
  - the document size: 111MB
  - number of XML nodes: 2,082,854
  - number of different tags: 376,906
- XPA utilized R-tree
- Queries:

Q1:/site/regions/africa/item[location='United']

Q2:/site/closed_auctions/closed_auction/annotation/description/parlist/listitem/parlist/listitem/text/emph/keyword/

# Experimental results

- XPA Q2 query evaluation

| Step | Nodes | Useful | Time [s] | DAC |
|---|---|---|---|---|
| site | 1 | 1 | 0.02 | 5 |
| closed_auctions | 1 | 1 | 0 | 5 |
| closed_auction | 9,750 | 9,750 | 1.9 | 1,386 |
| annotation | 9,750 | 9,750 | 4.6 | 50,594 |
| description | 9,750 | 2,934 | 5 | 50,252 |
| parlist | 2,934 | 2,934 | 4.64 | 49,773 |
| listitem | 8,512 | 1,713 | 1.9 | 15,448 |
| parlist | 1,713 | 1,713 | 4.02 | 43,114 |
| listitem | 4,964 | 4,964 | 1.02 | 8,872 |
| text | 4,964 | 1,890 | 2.11 | 24 999 |
| emph | 2,864 | 173 | 1.97 | 24,806 |
| keyword | 180 | 180 | 0.95 | 14,070 |
| Sum | 55,383 | 36,003 | 28.27 | 283,324 |

# Experimental results

- XPA Q1 query evaluation

| Step | Nodes | Useful | Time [s] | DAC |
|---|---|---|---|---|
| site | 1 | 1 | 0.02 | 5 |
| closed_auctions | 1 | 1 | 0 | 5 |
| closed_auction | 9,750 | 9,750 | 1.9 | 1,386 |
| annotation | 9,750 | 9,750 | 4.6 | 50,594 |
| description | 9,750 | 2,934 | 5 | 50,252 |
| parlist | 2,934 | 2,934 | 4.64 | 49,773 |
| listitem | 8,512 | 1,713 | 1.9 | 15,448 |
| parlist | 1,713 | 1,713 | 4.02 | 43,114 |
| listitem | 4,964 | 4,964 | 1.02 | 8,872 |
| text | 4,964 | 1,890 | 2.11 | 24 999 |
| emph | 2,864 | 173 | 1.97 | 24,806 |
| keyword | 180 | 180 | 0.95 | 14,070 |
| Sum | 55,383 | 36,003 | 28.27 | 283,324 |

# Experimental results

- Query Q1

| | XPA with R-trees | XPA with Signature R-trees | Multidimensional approach |
|---|---|---|---|
| Query processing time [s] | 3.48 | 4.26 | 0.26 |
| DAC | 5 716 | 6 954 | 2 604 |

- Query Q2

| | XPA with R-trees | XPA with Signature R-trees | Multidimensional approach |
|---|---|---|---|
| Query processing time [s] | 28.27 | 27.14 | 0.49 |
| DAC | 283 324 | 280 029 | 934 |

# Conclusion

- We compared XPA element-based approach and MDA path-based approach.
- Results show that path-based approach overcomes element-based approach.
- Query processing without extensive structural joins.

# References


- M. Krátký, J. Pokorný, V. Snášel: *Implementation of XPath Axes in the Multi-dimensional Approach to Indexing XML Data*. ***Accepted at International Workshop on Database Technologies for Handling XML information on the Web, DataX***, Int'l Conference on EDBT, Heraklion - Crete, Greece, 2004.
- M. Krátký, T. Skopal, and V. Snášel: ***Multidimensional Term Indexing for Efficient Processing of Complex Queries***. Kybernetika, Journal of the Academy of Sciences of the Czech Republic, 2004, accepted.
- T. Grust. ***Accelerating XPath Location Steps***. In Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, USA. ACM Press, June 4-6, 2002.