Schema-less, Semantics-based Change Detection for XML Documents

Shuohao Zhang
Curtis Dyreson
School of E.E. and Computer Science
Washington State University

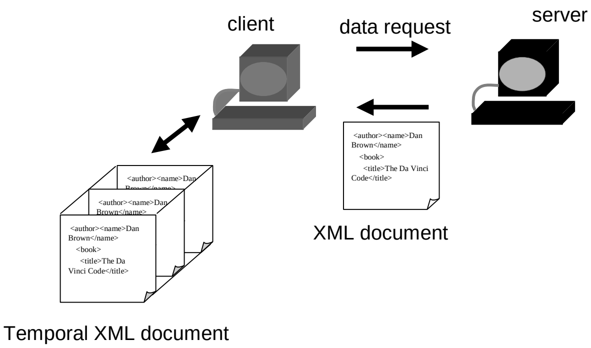
Richard T. Snodgrass
Department of Computer Science
University of Arizona

WISE 2004 - Brisbane

Outline

- Motivation
- Related work
- Our technique
- Experiments
- Conclusion

Target Application



Authors : Version 1

```
<author><name>Dan Brown</name>
   <book>
     <title>The Da Vinci Code</title>
     <publisher>Doubleday</publisher>
     <listprice>$24.95</listprice>
   </book>
   <book>
     <title>Angels & Demons</title>
     <publisher>Pocket Star</publisher>
   </book>
 </author>
```


Authors: Version 2

```
<author><name>Dan Brown</name>
   <book>
     <title>The Da Vinci Code</title>
     <publisher>Doubleday</publisher>
     <saleprice>$14.97</saleprice>
     <isbn>0385504209</isbn>
   </book>
   <book>
     <title>Angels & Demons</title>
     <publisher>Pocket Star</publisher>
     <listprice>$7.99</listprice>
   </book>
 </author>
```


Change Detection

- Edit script
- One deletion
  `<listprice>$24.95</listprice>`
- Two insertions
  `<saleprice>$14.97</saleprice>`
  `<isbn>0385504209</isbn>`
- Cost is # of inserts/updates/deletes

Authors: Version 3 (Very Different Struc.)

```
<author>
   <name>Dan Brown</name>
   <book>
     <title>The Da Vinci Code
     </title>
     <publisher>Doubleday
     </publisher>
     <listprice>$24.95
     </listprice>
   </book>
   <book>
     <title>Angels & Demons
     </title>
     <publisher>Pocket Star
     </publisher>
   </book>
 </author>
```

```
<publisher>Doubleday
   <book>
     <title>The Da Vinci Code</title>
     <author>
       <name>Dan Brown</name>
     </author>
     <listprice>$24.95</listprice>
   </book>
 </publisher>
 <publisher>Pocket Star
   <book>
     <title>Angels & Demons</title>
     <author>
       <name>Dan Brown</name>
     </author>
     <listprice>$7.99</listprice>
   </book>
 </publisher>
```


Related Work

- Text document change detection
  - D-band – Myers, *Algorithmica*, 1986
  - CVS
- Tree matching (Structure-based change detection)
  - Tree-correction – Wagner and Fischer. *JACM*, 1974
  - Ordered – Chawathe and Garcia-Molina, *SIGMOD* 1996
  - Unordered tree is NP-hard – Zhang et al., *IPL* 1992
  - HTML – Douglis and Ball, *USENIX* 1996
  - XML – Yang et al., Niagra project at UWisconsin 2004
  - XML – G. Cobéna, S. Abiteboul, A. Marian. *ICDE*, 2002 (Xyleme)
- Problem is different: Same information, very different structure

Outline

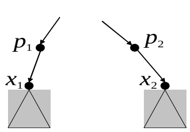
- Motivation
- Related work
- Our technique
- Experiments
- Conclusion

Our Approach

- Maxim: Information that *identifies* an element is conserved across changes to the element.
- Two step strategy
  1. Compute identifiers (assumption is no schema)
  2. Match an element based on identifying information

Data Model Node Semantics

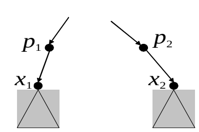
- Are x_1 and x_2 the same or different?



- **Axiom I**: *Nodes that are structurally different* (*modulo reordering*) *are semantically different*.
- If the blue triangles are different, then semantically different.

Data Model Node Semantics

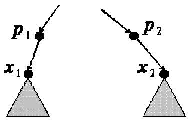
- x_1 and x_2 have the same structure, but are they duplicates or different?



- **Axiom II**: *Nodes that are structurally identical are semantically identical if and only if their respective parents are semantically identical, or if they are both root nodes*.
- If p_1 and p_2 are semantically different, then x_1 and x_2, though structural duplicates, are (contextually) different

Identifiers

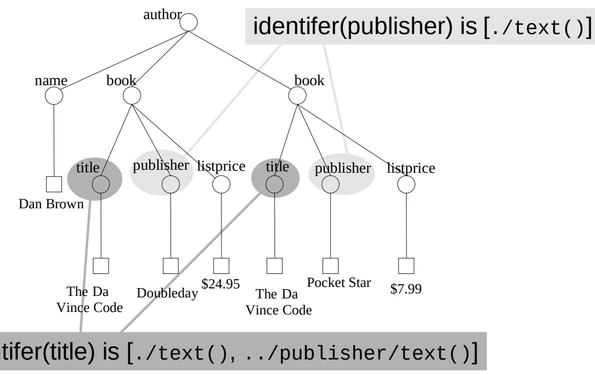
- *Type* is list of labels on path to element



  - Type is `p.x`
- *Abbreviated type*, last label, e.g., x
- A type identifier is a list of XPath expressions
- For any pair of type *T* nodes, *x* and *y* are semantically different if and only if

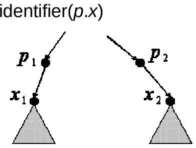
  Eval(*x*, *identifer*(*T*)) ≠ *Eval*(*y*, *identifer*(*T*)).

Identifier Example



Computing Identifiers

- Bottom up
  - Leaf is floor 0
- For each type at floor *k*
  - Choose identifier from among children – test
    - For type *p*, try identifier(*p.x*)



  - If *p* has structural duplicates then there will be no identifier.
    - Need parent's identifier in combination with identifier(*p.x*)
- Time complexity is *O*(*n**log(*n*))
- Space is *O*(*n*)

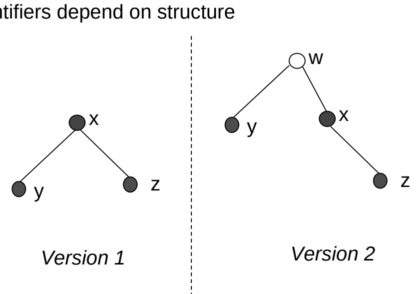
Authors and the Books They've Written

```
<bib>
  <author><name>n1</name>
    <book>
      <title>t1</title>
      <publisher>p1</publisher>
    </book>
  </author>
  <author><name>n2</name>
    <book>
      <title>t2</title>
      <publisher>p2</publisher>
    </book>
    <book>
      <title>t1</title>
      <publisher>p1</publisher>
    </book>
  </author>
</bib>
```

Type	Identifier
name	(`text()`)
author	(`name/text()`)
title	(`../../author/name/text(), text()`)
publisher	(`../../author/name/text(), text()`)
book	(`../author/name/text(), title/text()`)

Semantic Change Detection

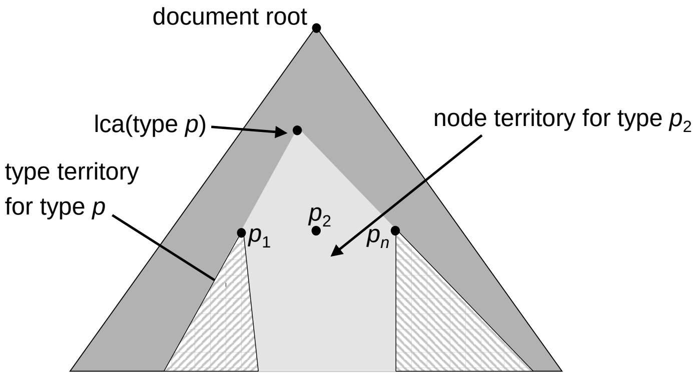
- *Assumption*: Identifying information is preserved across document changes
- Problem is structure could change
  - Identifiers depend on structure



Node and Type Territory

- Solution: Identifying information will remain *nearby*
- **Definition** [Type Territory] The territory of a type *T* is the set of all text nodes that are descendants of the least common ancestor of all of the type *T* nodes.
- Within the type territory is the territory controlled by individual nodes of that type.
- **Definition** [Node Territory] The territory of a type *T* node *p* is the type territory of *T* excluding all text nodes that are descendants of other type *T* nodes.

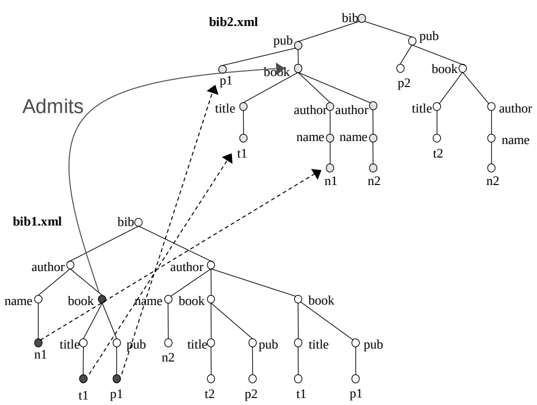
Node and Type Territory Depicted



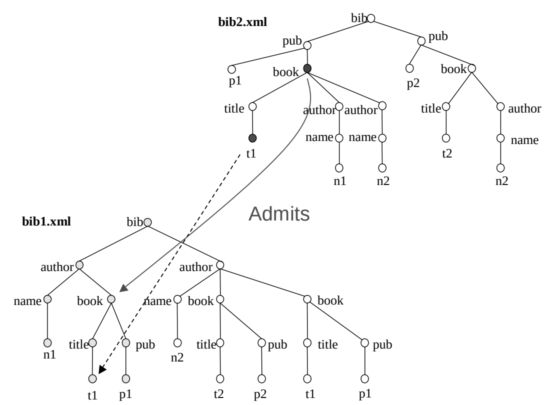
Matching

- **Definition** [Admits] *q admits p* if *Eval*(*q*, identifier(*q*)) is in the node territory of *p*.
- **Definition** [Node match] Nodes *p* and *q* are *matched* if and only if *p* and *q* admit each other.

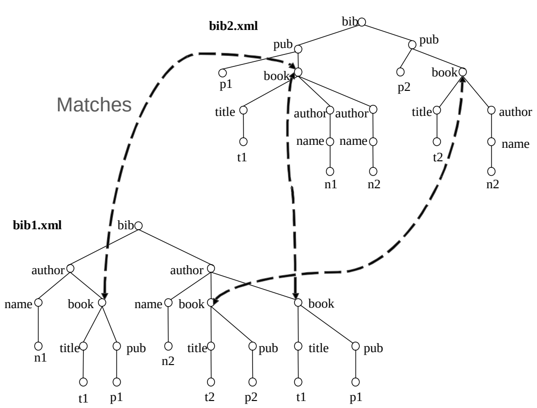
Example of Node Admittance



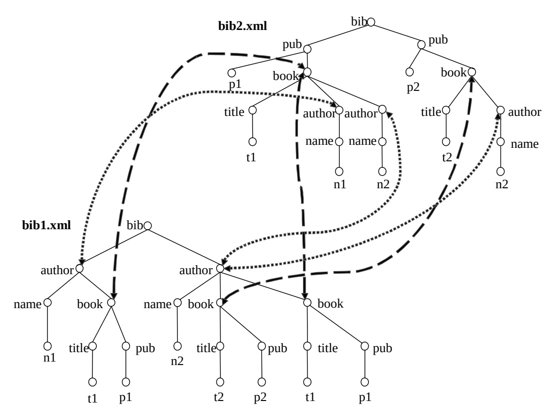
Example of Node Admittance



Book Matches



Book and Author Matches



Nature of Changes

- Complexity of technique
  - Time $O(n*\log(n)+p*\log(p))$ where n and p are # of nodes in early and later versions
  - Space $O(n+p)$
- Each match creates an association
  - Different structure -> new version of “same” node
  - Same structure -> extend lifetime of old version
- Unassociated nodes
  - In early version -> deleted from later version
  - In later version -> inserted into later version

Outline

- Motivation
- Related work
- Our technique
- Experiments
- Conclusion

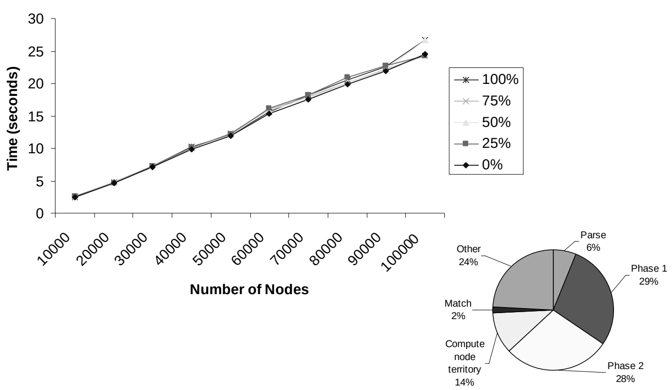
Experiments

- Goals
  - How does technique scale?
  - Examine cases where structural matching has problems
- Environment
  - PC with
    - hyperthreaded 2.8GHZ CPU
    - 2GB SDRAM
    - Windows XP
    - Java (jdk 1.4.2)
  - Isolated for experiments

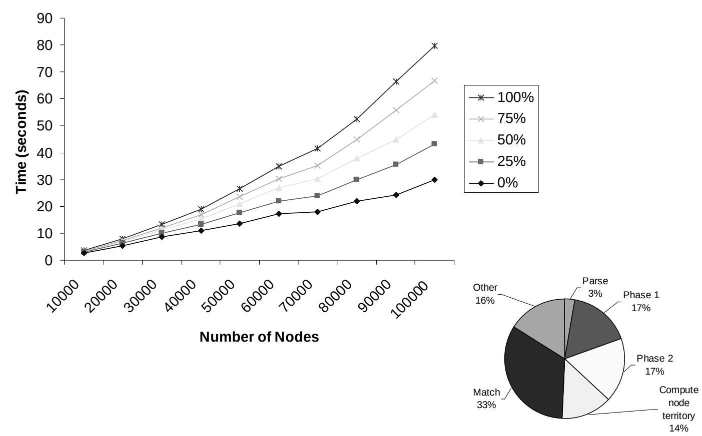
Experiments (continued)

- Methodology
  - Choose first XQuery use case – author/publisher/book
  - Randomly generate documents increasing in size, from 10,000 to 100,000 elements
  - Test 0% to 100% match percentage
  - Measure time
  - Average over several runs
- Experiment 1: permute ordering of elements, but same structure
  - Structural change detection based on ordered trees would fail
- Experiment 2: same information, different structure
  - Structural change detection can't match

Match Different Orderings



Match Different Structures



Conclusion

- New change detection technique
  - Problem: same information, very different structure
  - Information that identifies an element is conserved across changes
- Some experiments
  - Practical for low-update, in-memory systems
  - Target application: Temporal XML databases
- Future work
  - Integrate with Apache web server (HTML-based, Dyreson et al. WWW 2004)
  - Integrate with temporal query languages (tauXQuery, Gao and Snodgrass, VLDB 2003; *TT*XPath, Dyreson, WISE 2001)
  - Utilize schema keys
  - Persistence