Accessing TEI XML Documents

James Cummings *Date: November 2004*

1 Methods of Access

So you've created some TEI XML documents, what now?

- XPath
- XML Query (XQuery)
- XSLT Tranformation another format (HTML, PDF, RTF, CSV, etc.)
- Custom Applications (Xaira, TEIPubisher, Philologic etc.)

2 What is XPath?

- It is a syntax for accessing parts of an XML document
- It uses a path structure to define XML elements
- It has a library of standard functions
- It is a W3C Standard
- It is one of the main components of XQuery and XSLT

3 XPath: You Already Understand Paths

Assume we have a document that solely contains:

```
<person sex="m" age="78">
  <persName>
    <foreName>Hans</foreName>
    <surname>von Hülsen</surname>
  </persName>
  <birth date="1890-04-05"/>
  <death date="1968-04-14"/>
  <nationality code="Z_"/>
</person>
```

4 XPath: What do these paths get us?

- /person/persName/surname
- /person/@sex
- /person/birth/@date
- /person/persName/foreName/../../death/@date
- //surname
- persName/foreName

If an XPath starts with a / then it is an absolute path to an element. If an XPath starts with // then all elements in the document that fulfill the criteria will be selected

5 XPath: Unknown Elements

- /person/*/surname
- //persName/*
- /person/nationality/@*
- /person/*
- //*

6 XPath: Branches of the Tree

- /person/persName[1]
- /person/persName/*[1]
- /person/persName[surname]
- /person/persName/*[last()]
- //person[@age > 70]
- //person[@age]

Terminating an XPath with something in square brackets allows you to branch or filter that path based on the next element.

7 XPath: Selecting Multiple Paths

- /person/persName/surname | /person/nationality
- //surname | //birth | //death
- //nationality/@code | //surname | //@sex

8 XPath: More About Paths

- A location path results in a node-set
- Paths can be absolute (/person/persName)
- Paths can be relative (persName/surname)

Formal Syntax:

```
axisname::nodetest[predicate]
```

for example:

```
child::persName[foreName = 'Hans']
```

9 XPath: Axis

ancestor	Contains all ancestors (parent, grandparent, etc.) of the current node
ancestor-or-self	Contains the current node plus all its ancestors (parent, grandparent, etc.)
attribute	Contains all attributes of the current node
child	Contains all children of the current node
descendant	Contains all descendants (children, grandchildren, etc.) of the current node
descendant-or-self	Contains the current node plus all its descendants (children, grandchildren, etc.)
following	Contains everything in the document after the closing tag of the current node
following-sibling	Contains all siblings after the current node
parent	Contains the parent of the current node
preceding	Contains everything in the document that is before the starting tag of the current node
preceding-sibling	Contains all siblings before the current node
self	Contains the current node

10 XPath: axis::nodetest Examples

ancestor::person	Selects all the 'person' ancestors of the current node.
ancestor-or-self::person	Selects the current node plus all 'person' ancestors.
attribute::age	Selects the age attribute of the current node.
child::surname	Selects the surname child of the current node
descendant::person	Selects all the person descendants of the current node
descendant-or-self::persName	Selects current node and all persName descendants
following::person	Selects all person elements after the current node
following-sibling::person	Selects all person siblings after the current node

parent::persName	Selects the persName parent element of the curent node
preceding::person	Selects the person element that is preceding the current node.
preceding-sibling::person	Selects the preceding person elements that are all siblings before the current node
self::person	Selects the current person node

11 XPath: Predicates

- child::person[attribute::age='74']
- child::person[@age='74']
- child::person[position()='1']
- child::person[1]
- child::person[last()]
- child::person[last()-1]

12 XPath: Abbreviated Syntax

nothing	child::	person is short for child::person	
@	attribute::	@age is short for attribute::age	
.	self::node()	./birth is short for self::node()/birth	
..	parent::node()	../birth is short for parent::node()/child::birth	
//	descendant-or-self::node()	person//surname is short for child::person/descendant-or-self::node()/child::surname	

13 XPath: Operators

XPath has support for numerical, equality, relational, and boolean expressions

+	Addition	3 + 2	= 5
-	Subtraction	10 - 2	= 8
	Multiplication	6 * 4	= 24
div	Division	8 div 4	= 2
mod	Modulus	5 mod 2	= 1
=	Equal	@age = '74'	True (if @age does equal '74')
!=	Not equal	@age != '74'	False
<	Less than	@age < '84'	True
<=	Less than or equal	@age <= '72'	False
>	Greater than	@age > '25'	True
>=	Greater than or equal	@age >= '72'	True
or	Boolean OR	@age = '74' or @age = '64'	True
and	Boolean AND	@age <= '84' and @age = '74'	True

14 XPath Functions: Node-Set Functions

count()	Returns the number of nodes in a node-set	count(person)
id()	Selects elements by their unique ID	id('S3')
last()	Returns the position number of the last node	person[last()]
name()	Returns the name of a node	//*[name('person')]
namespace-uri()	Returns the namesparce URI of a specified node	namespace-uri(persName)
position()	Returns the position in the node list of the node that is currently being processed	//person[position()='6']

15 XPath Functions: String Functions

concat()	Concatenates its arguments	concat('http://', $domain, '/', $file, '.html')
contains()	Returns true if the second string is contained withing the first string	//persName[contains(surname, 'van')]

normalize-space()	Removes leading and trailing whitespace and replaces all internal whitespace with one space	normalise-space(surname)
starts-with()	Returns true if the first string starts with the second	starts-with(surname, 'van')
string()	Converts the argument to a string	string(@age)
substring	Returns part of a string of specified start character and length	substring(surname, 5,4)
substring-after()	Returns the part of the string that is after the string given	substring-after(surname, 'De')
substring-before	Returns the part of the string that is before the string given	substring-before(@date, '-')
translate()	Performs a character by character replacement. It looks at the characters in the first string and replaces each character in the first argument given for the one in the same position in the string2	translate('1234', '24', '68')

16 XPath Functions: Numeric Functions

ceiling()	Returns the smallest integer that is not less that the number given	ceiling(3.1415)
floor()	Returns the largest integer that is not greater than the number given	floor(3.1415)
number()	Converts the input to a number	number('100')
round()	Rounds the number to the nearest integer	round(3.1415)
sum()	Returns the total value of a set of numeric arguments	sum(//person/@age)
not()	Returns true if the condition is false	not(position() >5)

17 XPath: Where can I use XPath?

Learning all these functions, though a bit tiring to begin with can be very useful as they are used throughout XML technologies, but especially in XSLT and XQuery.

18 What is XQuery?

- It is a domain-specific method for accessing and manipulating XML
- It is meant for querying XML
- It is built upon XPath
- It is like SQL but for XML
- A W3C proposed recommendation

19 XQuery: Expressions

path expressions return a nodeset

element constructors return a new element

FLWOR expressions analogous to SQL Select statement

list expressions operations on lists or sets of values

conditional expressions traditional if then else construction

qualified expressions boolean operations over lists or sets of values

datatype expressions test datatypes of values

20 XQuery: Path Expression

The simplest kind of XQuery that you've already seen:

```
document("test.xml")//p

//p/foreign[@lang='lat']

//foreign[@lang='lat']/text()
```

21 XQuery: Element constructor

May contain literal text or variables:

```
<latin>o tempora o mores</latin>

<latin>{$s}</latin>
```

22 XQuery: FLWOR expressions

For - Let - Where - Order - Return

```
for  $t in //text
let $lats := $t//foreign[@lang='lat']
where count($lats) > 1
order by count($lats)
return
<latin>
{$lats}
<txt>{$t/@id}</txt>
</latin>
```

- `for` defines a cursor over an xpath
- `let` defines a name for the contents of an xpath
- `where` selects from the nodes as in SQL
- `order` sorts the results as in SQL
- `return` specifies the XML fragments to be constructed
- Curly braces are used for grouping, and define the scope of the for clause
- This is one of the most common forms of XQuery, and is often used for the equivalent of SQL joins.

23 XQuery: List Expressions

XQuery expressions manipulate lists of values, for which many operators are supported:

- constant lists: (7, 9, <thirteen/>)
- integer ranges: i to j
- XPath expressions
- concatenation
- set operators: | (or union), intersect, except
- functions: remove, index-of, count, avg, max, min, sum, distinct-values ...

When lists are viewed as sets:

- XML nodes are compared on node identity
- duplicates are removed
- the order is preserved

24 XQuery: Conditional Expressions

```
<div>
   {
     IF document("xqt")//title/text()
       ="Introduction to XQuery"
     THEN <p>This is true.</p>
     ELSE <p>This is false.</p>
   }
</div>
```

25 XQuery: Qualified Expressions

- some-in-satisfies

```
for $b in document("book.xml")//text
where some $p in $b//p satisfies
  (contains($p,"sailing") AND contains($p,"windsurfing"))
return $b/ancestor::teiHeader//title[1]
```

- every-in-satisfies

```
for $b in document("book.xml")//text
where every $p in $b//p satisfies
  contains($p,"sailing")
return $b/ancestor::teiHeader//title[1]
```

26 XQuery: Datatype Expressions

- XQuery supports all datatypes from XML Schema, both primitive and complex types
- Constant values can be written:
  - as literals (like string, integer, float)
  - as constructor functions (true(), date("2001-06-07"))
  - as explicit casts (cast as xsd:positiveInteger(47))
- Arbitrary XML Schema documents can be imported into an XQuery
- An `instance of` operator allows runtime validation of any value relative to a datatype or a schema.
- A `typeswitch` operator allows branching based on types.

27 eXist: Looking for words

We are going to be using the eXist native XML Database to practice our XQueries. It has some useful text searching capabilities. For example:

```
//p &= 'fish dutch'
```

will find paragraphs containing both the words `fish` and `dutch` (in either order), and is rather easier to type than the equivalent xpath:

```
//p[contains(.,'fish') and contains(.,'dutch')]
```

In eXist you can also do a proximity search:

```
//p[near(.,'fish dutch',20)]
```

as well as stem matching:

```
//p &= 'fish*'
```

28 Choosing between XML Technologies

- Not locked into one solution
- Multiple technologies work well together
- Different technologies for different purposes

29 A choice of generic XML vocabularies

- XML Schema: describes structures and data types;
- XPath: describes how to address any part of an XML document
- XSLT: describes how to transform an XML document;
- XQuery: an XML database query language.

30 Storage strategies

Data has to be stored somewhere. How should XML data be managed? There are several possibilities:

1. as discrete XML documents
2. within any convenient DBMS
3. within an XML fragment repository or native XML Database

31 DBMS or XML?

Do you have to choose?

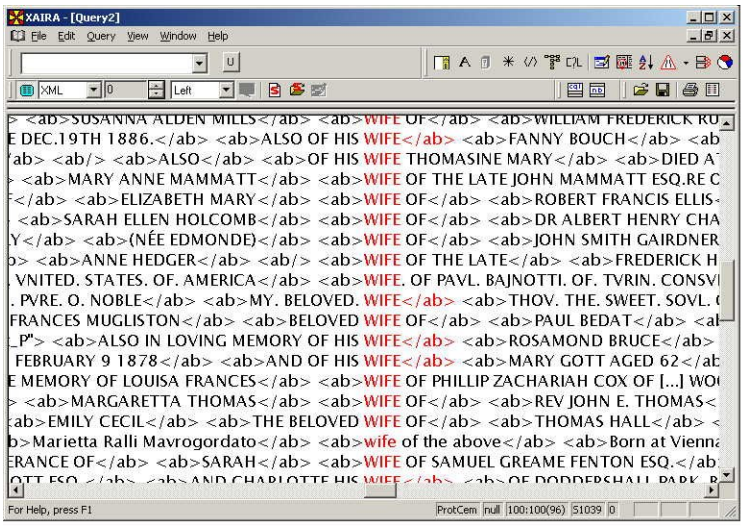
- The argument from history
  1. flatfiles gave way to network DBMS
  2. network DBMS gave way to relational
  3. will relational DBMS give way to XML databases?
- Getting the best of both worlds
  - DBMS are good at storing and managing relations
  - but equivalent technologies for XML are rapidly maturing
  - and even XML support is present in many DBMS
  - for XML documents, an XML solution is best

32 Other Related Technologies

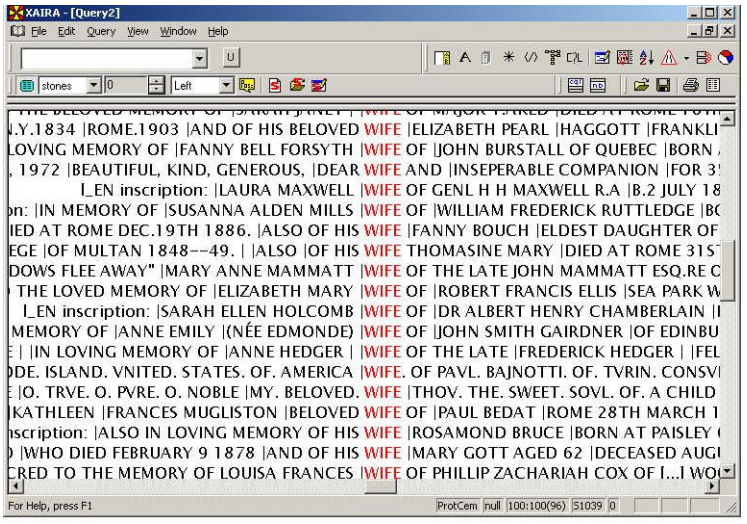
- TEIPublisher
- Apache's Cocoon
- Philologic
- Leaders
- Xaira

33 stone-wife-0

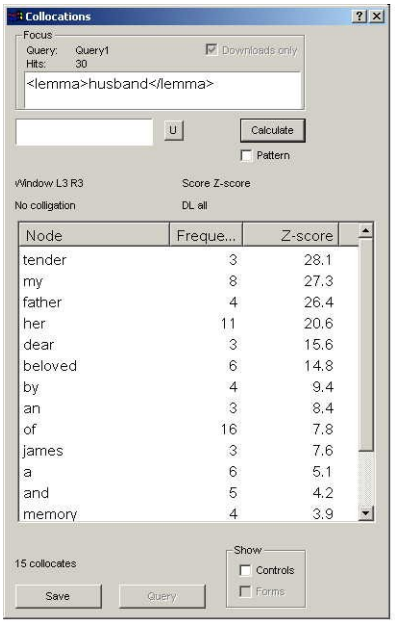
34 stone-wife1



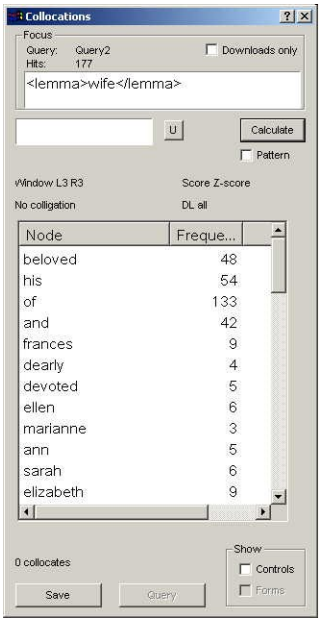
35 stone-wife2



36 stones-husb-collox



37 stones-wife-collox



38 Delivery strategies

- Our goal is fast and efficient access to any subtree of the docuverse, of any size
- Xpath has an adequately rich semantics
- XSLT has an adequately rich syntax
- XQuery offers all the programming features we need
- The rest is a Simple Matter of Programming...