

XML databases

Code: xml-db

Author and version

- Daniel K. Schneider
- Email: Daniel.Schneider@tecfa.unige.ch
- Version: 0.1 (modified 25/11/07 by DKS)

Prerequisites

- HTML
- Some knowledge of XML and XSLT would be better



Objectives

- Understand the roles of XML as data format
- Understand the role and concept of an XML databases
- Master basics of the XQuery language

1. Table of Contents

1. Table of Contents	2
2. Introduction	4
2.1 Some XML in the three tier architecture model	5
2.2 XML document workflows	6
3. Recall of XML principles	7
Example 3-1:XHTML Page	7
3.1 “Well-formed” XML documents	8
Example 3-2:A minimal well-formed XML document	9
3.2 Valid XML documents	10
3.3 Document type definitions (DTDs)	11
A.Understanding DTDs by example	12
Example 3-3>Hello text with XML	12
B.Summary syntax of element definitions	13
3.4 Types of XML languages	14
4. Recall of XPath	15
5. What is XQuery ?	16
5.1 Principle	16
5.2 Introductory examples	17
5.3 Use contexts	18
6. Basic XQuery	19
6.1 The query expression	19
6.2 FLWOR expressions	20
A.For	20
B.let	21
C.where	21
D.order	22
E.return	23
6.3 Building XML fragments	24
A.Multi-fragment version (collection)	24

B.Version single fragment 25

6.4 For within for

26

2. Introduction

XML can play multiple roles with respect to web databases

(1) As communication format

- Data can be XML before they are put into a database (insures data integrity)
- Output from database can be transferred in XML format (separates content from form)
- Servers can communicate via XML communication languages (e.g. SOAP)

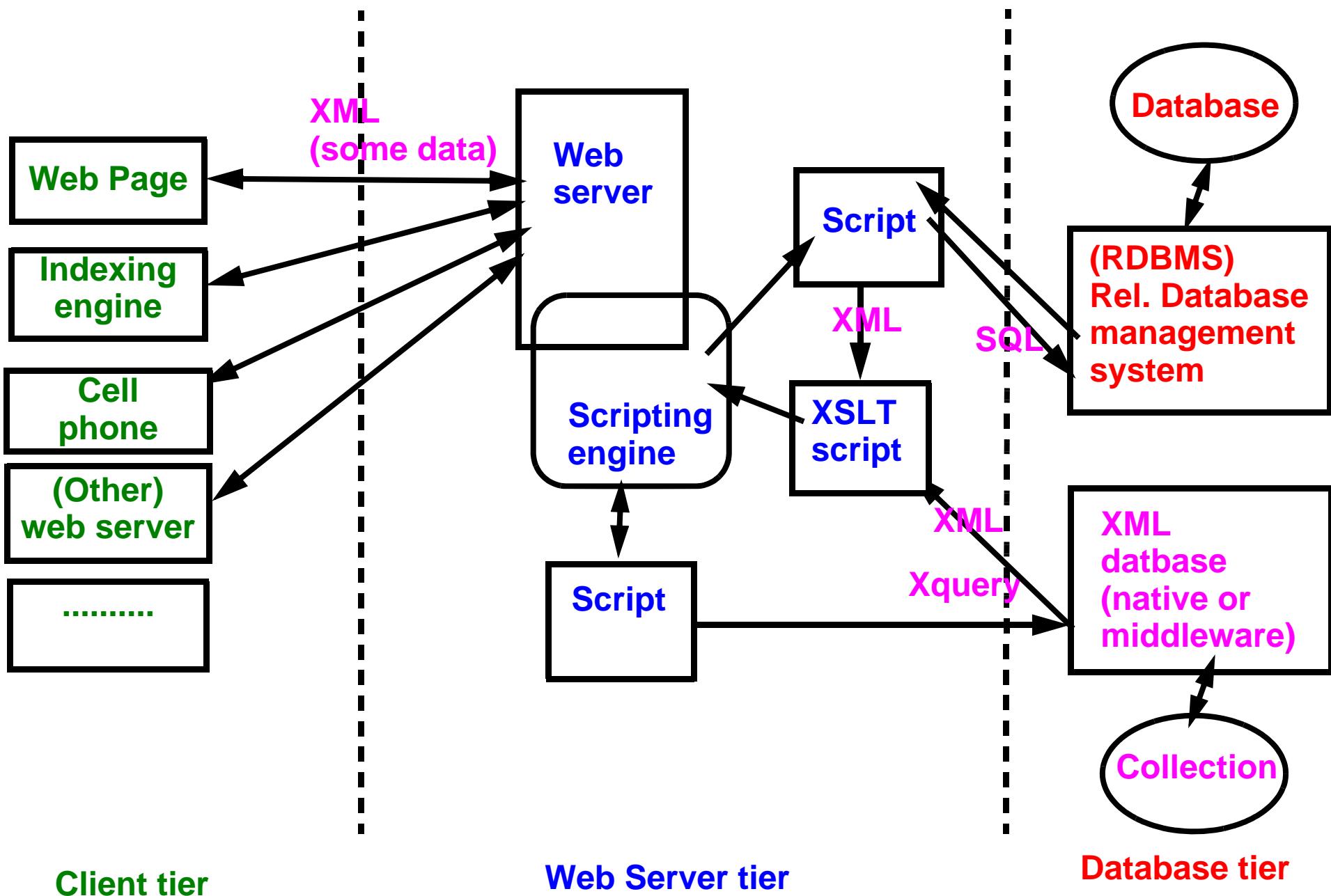
(2) As data format

- XML files can be considered a very simple form of database (e.g. a long list of references)
- XML can be stored in relation databases, e.g. some short elements in fields to be indexed and the rest in large blobs.
- XML can be stored as XML in a database, usually an XML database

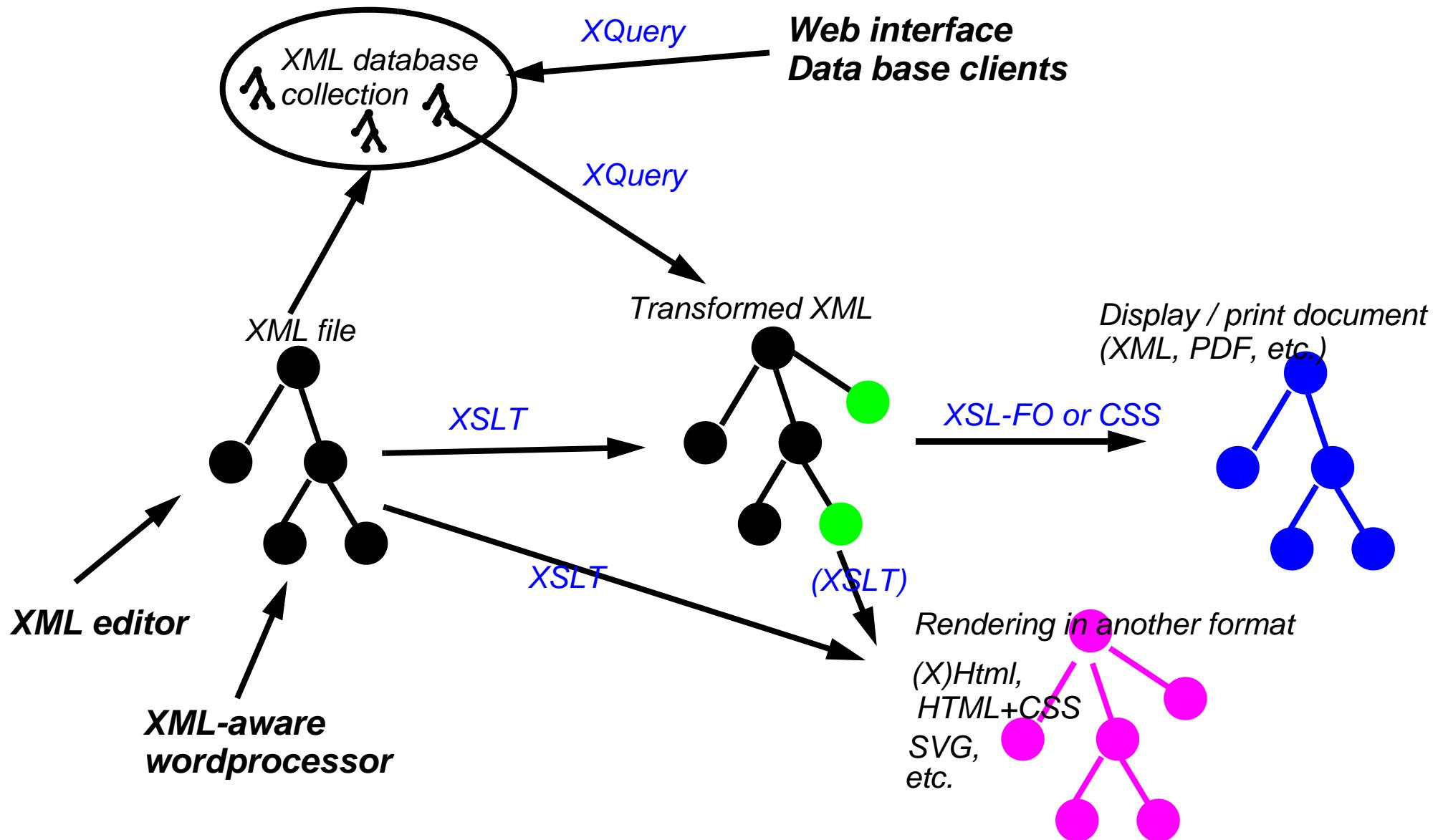
Native XML databases

- Native XML databases can preserve physical structure (entity usage, CDATA sections, etc.) as well as comments, PIs, DTDs, etc.
- Native XML databases can store XML documents without knowing their schema (DTD), assuming one even exists.
- The only interface to the data in native XML databases is XML and related technologies (such as XQuery, XPath, the DOM) or an XML-specific API, such as the XML:DB API. XML-enabled databases, on the other hand, offer direct access to the data, such as through ODBC.

2.1 Some XML in the three tier architecture model



2.2 XML document workflows



3. Recall of XML principles

Example 3-1: XHTML Page

- XHTML is an XML application (i.e. it uses the XML formalism).
- Some XML principles:
 - tags (and a way to combine them), a single root, ...
 - XML declaration on top
 - a “grammar” (DTD), namespace declaration, ...

```
<?xml version = "1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns = "http://www.w3.org/1999/xhtml">
  <head>
    <title>Object Model</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  </head>
  <body>
    <h1>Welcome to our Web page!</h1>
    <p>Enjoy ! </p>
  </body>
</html>
```

XML and Document type declaration

Namespace declaration

Encoding declaration

Body (page contents)

3.1 “Well-formed” XML documents

- A document must start with an **XML declaration** (including version number !)

```
<?xml version="1.0"?>
```

- You may specify encoding (default is utf-8) and you have to stick to an encoding !

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

- Structure must be **hierarchical**:

- start-tags and end-tags must match

- no cross-overs. E.g. this is bad: <i>......</i>

- case sensitivity, e.g. "LI" is not "li"

- "EMPTY" tags must use "self-closing" syntax:

- e.g.
</br> should be written as
, a lonely "
" would be illegal

- Attributes must **have values** and values are quoted:

- e.g. or <person status="employed">

- e.g. <input type="radio" checked="checked">

- A **single root element** per document

- Root element opens and closes content

- The root element should not appear in any other element

- Special characters (!!): <, &, >, ", '

- Use < & > " ' instead of <, &, >, ", '

- Applies also to URLs !!

bad: http://truc.unige.ch/programme?bla&machin

good: http://truc.unige.ch/programme?bla&machin

Example 3-2: A minimal well-formed XML document

```
<?xml version="1.0" ?>
<page updated="jan 2007">
  <title>Hello friend</title>
  <content> Here is some content :) </content>
  <comment> Written by DKS/Tecfa </comment>
</page>
<hello> Hello <important>dear</important> reader ! </hello>
```

- It has an XML declaration on top
- It has a root element (i.e. **page**)
- Elements are nested and tags are closed
- Attribute has quoted value

XML names and CDATA Sections

- Names used for elements should start with a letter and only use letters, numbers, the underscore, the hyphen and the period (no other punctuation marks) !
 - Good: <**driversLicenceNo**> <**drivers_licence_no**>
 - Bad: <**driver's_licence_number**> <**driver's_licence_#**> <**drivers licence number**>
- When you want to display data that includes "XMLish" things that should not be interpreted you can use so called CDATA Sections:

```
<example>
  <!CDATA[ (x < y) is an expression
            <svg xmlns="http://www.w3.org/2000/svg">
        ]]> </example>
```

3.2 Valid XML documents

An valid document must be:

1. “well-formed” (see above)
2. **conform to a grammar**, .e.g.
 - only use tags defined by the grammar
 - respect nesting, ordering and other constraints

Kinds of XML grammars

1. DTDs are part of the XML standard. Most grammars are written with DTD.
2. **XML Schema** is a more recent W3C standard, used to express stronger constraints
3. **Relax NG** is a OASIS standard (an alternative to XML Schema ...)
4. **Schematron** (yet another alternative, constraint-based)

3.3 Document type definitions (DTDs)

- A **valid** XML document includes a **declaration that specifies the DTD** used
- DTD is declared on top of the file after the XML declaration.
- XML declarations, DTD declaration etc. are part of the prologue
- So: The <!DOCTYPE...> declaration is part of the XML file, **not** the DTD

Example:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE hello SYSTEM "hello.dtd">
<hello>Here we <strong>go</strong> ... </hello>
```

4 ways of using a DTD

1. No DTD (XML document will just be well-formed)
2. DTD rules are defined inside the XML document
 - We get a "standalone" document (the XML document is self-sufficient)
3. "Private/System" DTDs, the DTD is located on the system (own computer or the Internet)
 - ... **that's what you are going to use when you write your own DTDs**

```
<!DOCTYPE hello SYSTEM "hello.dtd">
```
4. Public DTDs, we use a name for the DTD.
 - means that both your XML editor and user software know the DTD
 - strategy used for common Web DTDs like XHTML, SVG, MathML, etc.

Note: There are other Schemas than DTDs. Association with XML doesn't work the same way!

A. Understanding DTDs by example

- Recall that DTDs define all the elements and attributes and the way they can be combined

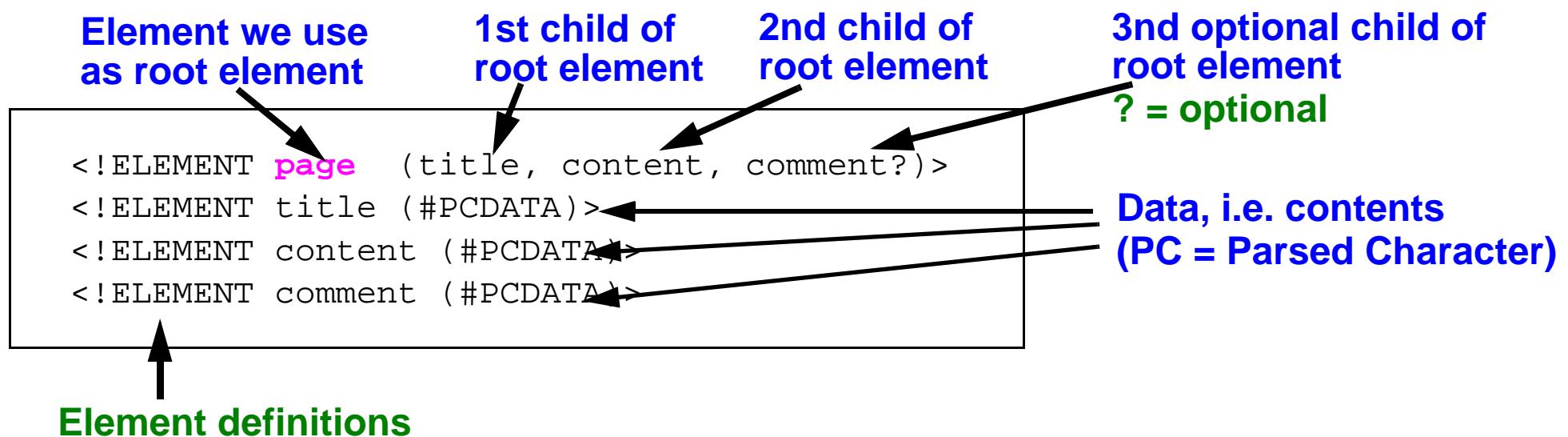
Example 3-3: Hello text with XML

url: init/very-simple-page.xml

A simple XML document of type <page>

```
<page>
  <title>Hello friend</title>
  <content>Here is some content :)</content>
  <comment>Written by DKS/Tecfa, adapted from S.M./the Cocoon samples</comment>
</page>
```

A DTD that would validate the document



B. Summary syntax of element definitions

- The purpose of this module is not to teach you how to write DTDs
- To understand how to use DTDs, you just need to know how to read a DTD

syntax element	short explanation	Example Element definitions Valid XML example
,	• elements in that order	<!ELEMENT Name (First, Middle, Last)> • Element Name must contain First, Middle and Last <pre><Name> <First>D.</First><Middle>K.</Middle><Last>S.</Last> </Name></pre>
?	• optional element	<!ELEMENT Name (First,Middle?,Last)> • Middle is optional <pre><Name><First>D.</First><Last>S.</Last></Name></pre>
+	• at least one element	<!ELEMENT list (movie+) <list><movie>Return of ...</movie> <movie>Comeback of ...</movie> </list>
*	• zero or more elements	<!ELEMENT list (item*)> • almost as above, but list can be empty
	• pick one (or operator)	<!ELEMENT major (economics law) <major> <economics> </economics> </major>
()	• grouping construct, e.g. one can add ? or * or + to a group.	<!ELEMENT text (para list title)*> <text> <title>Story</title><para>Once upon a time</para> <title>The awakening</title> <list> ... </list> </text>

3.4 Types of XML languages

XML-related languages can be categorized into the following classes:

1. XML accessories, e.g. XML Schema
 - Extend the capabilities specified in XML
 - Intended for wide, general use
 - Examples: XML Schema, RDF, ...
2. XML transducers: e.g. XSLT
 - Converts XML input data into output
 - Associated with a processing model
 - Examples: XSLT, XPath, XQuery, Xlink
3. XML applications, e.g XHTML
 - Defines constraints for a class of XML data
 - Intended for a specific application area
 - Dataformats like: XHTML, RSS, SVG, Docbook, MathML
 - Communication languages like: SOAP

4. Recall of XPath

Syntax element	(Type of path)	Example path	Example matches
name	child element name	project	<project> </project>
/	child / child	project/title	<project><title> ... </title>
		/	(root element)
//	descendant	project//title	<project><problem><title>....</title>
		//title	<root>...<title>..</title> (any place)
*	"wildcard"	*/title	<bla><title>..</title> and <bli><title>...</title>
	"or operator"	title head	<title>...</title> or <head> ...</head>
		* / @*	All elements: root, children and attributes
.	current element	.	
..	parent element	../problem	<project>
@attr	attribute name	@id	<xyz id="test">...</xyz>
element/@attr	attribute of child	project/@id	<project id="test" ...> ... </project>
@attr='value'	value of attribute	list[@type='ol']	<list type="ol"> </list>
position()	position of element in parent	position()	
last()	number of elements within a context	last() position() != last()	

- XPath is a language for addressing parts of an XML document
- In support of this primary purpose, it also provides basic facilities for manipulation of strings, numbers and booleans.

5. What is XQuery ?

5.1 Principle

- XQuery is an XML language to query XML
- Document types: single documents or collections of:
 - files
 - XML databases
 - XML fragments in memory (e.g. DOM trees)
- Relies on the Xpath (version 2.0) language
- Can generate new XML documents
- XQuery 1 and 2.2 does not define updating, see XQupdate (XML Query Updating)

Summary:

Xquery, XML Query Language, is a query language that uses the structure of XML intelligently can express queries across all these kinds of data, whether physically stored in XML or viewed as XML via middleware. In other words, you can use it to retrieve things from files, from XML representations made from SQL databases, from native XML databases like eXist. In addition, Xquery is quite a real programming language

Standards:

url: <http://www.w3.org/TR/xquery/> (**query, stable**)

url: <http://www.w3.org/TR/xqupdate/> (**updates, recent**)

5.2 Introductory examples

Find all nodes with path //topic/title

(returns all fragments `<topic><title>xxxx </title></topic>` from anywhere in the tree)

```
for $t in //topic/title  
    return $t
```

Find all nodes of //topic/title in "catalog.xml"

```
for $t in document("catalog.xml")//topic/title  
    return $t
```

same, but via HTTP

```
for $t in document("http://tecfa.unige.ch/proj/seed/catalog/net/  
catalog.xml")//topic/title  
    return $t
```

Result

```
<title>TECFA Seed Catalog</title>  
<title>Introduction</title>  
<title>Conceptual and technical framework</title>  
<title>The socio-constructivist .... etc.
```

5.3 Use contexts

- You need an XQuery processor
 - included in some command line/library products like “saxon”
 - included in XML databases.
 - included in an XML editor
 - included in some programming languages

Same principle as for SQL

- From the command line
- With an administration tool (Web or local)
- Through an other web application)

6. Basic XQuery

6.1 The query expression

Recall: XQuery uses XPath elements

1. position in a tree

- ex: returns all nodes <card-propvalue> under <c3mssoft-desc> ...

```
//c3msbrick//c3mssoft-desc//card-propvalue
```

2. comparison

- ex: returns all nodes with this id

```
//c3msbrick//c3mssoft[@id="epbl"]
```

3. functions

- ex: only returns the content of a node

```
//c3msbrick/title/text()
```

6.2 FLWOR expressions

- FLOWR = "For-Let-Where-Order-Return"
- Similar to select-from-where-..-order-by of SQL

Overview:

1. for = iteration on a list of XML nodes
2. let = binding of a result to a local variable
3. where = selection condition
4. order = sorting
5. return = expression that will be returned

A. For

Syntax: `for $variable in expression_search
RETURN $variable`

"for" associates to \$variable each XML fragment found with the XPath (see page 19)

Example:

```
for $t in //topic/title return $t
```

B. let

- "let" assigns a value (node) to a variable

- Example without let:

```
for $t in document("catalog09.xml")//c3msbrick//c3mssoft-desc//card-propvalue  
return $t
```

- Same example with let

- for each node \$t found, we bind \$desc with a subnode.

```
for $t in document("catalog09.xml")//c3msbrick  
  let $desc := $t//c3mssoft-desc//card-propvalue  
return $desc
```

- Counting, for each node \$t found in the "for" loop we count the number of subnodes:

```
for $t in document("catalog09.xml")//c3msbrick  
let $n := count($t//c3mssoft)  
return <result> {$t/title/text()} owns {$n} bricks </result>
```

C. where

- defines a selection condition

- Same as above, but we only return results with at least 2 <c3mssoft>

```
for $t in document("catalog09.xml")//c3msbrick  
  let $n := count($t//c3mssoft)  
  where ($n > 1)  
return <result> {$t/title/text()} owns {$n} bricks </result>
```

D. order

- Can sort results
- Alphabetic sorting:

```
for $t in //topic/title order by $t return $t
```

- Alphabetic sorting

```
for $t in document("catalog09.xml")//c3msbrick  
let $n := count($t//c3msssoft)  
where ($n > 1)  
order by $n  
return <result> {$t/title/text()} owns {$n} bricks </result>
```

- Slightly more complex return clause, we also include <titles> of <c3msbricks>:

```
for $t in document("catalog09.xml")//c3msbrick  
let $brick_softs := $t//c3msssoft  
let $n := count($brick_softs)  
where ($n > 0)  
order by $n  
return <result>  
    For {$t/title/text()} we found {$n} softwares:  
        {$brick_softs//title}  
</result>
```

E. return

- builds the expression to return
- Warning: Each iteration must return a fragment, i.e. a single node (and not a collection!)

Good:

```
for $t in document("catalog09.xml")//c3msbrick
let $n := count($t//c3msssoft)
return <result>
    {$t/title/text()} owns {$n} bricks
</result>
```

Bad:

```
for $t in document("catalog09.xml")//c3msbrick
let $n := count($t//c3msssoft)
return $t/title/text() owns $n bricks
```

6.3 Building XML fragments

- Result of an xquery usually should lead to a single node (not a list)

A. Multi-fragment version (collection)

Consider the following expression

```
for $t in //c3msbrick/title
    return $t
```

It will return a list (a collection)

```
1  <title class ="- topic/title " > TECFA Seed Catalog </ title >
2  <title class ="- topic/title " > Introduction </ title >
3  <title class ="- topic/title " > Conceptual and technical framework </
title >
4  <title class ="- topic/title " > The socio-constructivist approach </
title >
....
```

- This sort of list is not well-formed XML, and can not be displayed in a browser for example.
- ... but this is not a problem if is dealt with by some program (e.g. a php script querying a DOM tree)

B. Version single fragment

The following expression:

```
<result>
  { for $t in //topic/title/text()
    return <titre>{$t}</titre> }
</result>
```

returns:

```
<result  >
<titre > TECFA Seed Catalog </ titre >
<titre > Introduction </ titre >
<titre > Conceptual and technical framework </ titre >
<titre > The socio-constructivist approach </ titre >
.....
</result>
```

- We replaced "title" tags by "titre" tags (just to make this a bit more complicated)
- All expression within { ... } are evaluated
- but all <tags> are copied as is ...

6.4 For within for

- you may have loops within loops...

```
<result>
<title>List of C3MSBricks and associated Software</title>
{ for $t in document("catalog09.xml")//c3msbrick
  let $brick_softs := $t//c3mssoft
  let $n := count($brick_softs)
  where ($n > 0)
  order by $n descending
  return
    <brick> Pour {$t/title/text()} on a les {$n} modules suivants:
    { for $soft in $brick_softs
      return <soft> {$soft//title/text()}</soft>
    }
  </brick>
}
</result>
```

- Each FLWOR expression is within { ... }
- The "return" clause of the outer loop includes a loop that will deal with
 - \$brick_softs contains a collection of \$softs from which we extract titles
- We also sort the results

Result:

```
<result>
<title> List of C3MSBricks and associated Software </title>
<brick> Gallery owns les 5 bricks suivants:
<soft> PhotoShare </soft>
<soft> Photoshare combined with PageSetter </soft>
<soft> My_eGallery </soft>
<soft> Coppermine </soft>
<soft> Gallery </soft>
</brick>
<brick> User statistics owns les 5 bricks suivants:
<soft> commArt </soft>
<soft> pncUserPoints </soft>
<soft> pncSimpleStats </soft>
<soft> Statistics module </soft>
<soft> NS-User_Points </soft>
</brick>
.....
</result>
```

