# Chapter 1: Semistructured Data Management
# XML



The Web has generated a new class of data models, which are generally summarized under the notion semi-structured data models. The reasons for that are manifold: the ubiquity of documents and messages, the extensive use of the hypertext paradigm, the lack of schema information. In this chapter we want to explore their properties and techniques required to deal with them.

In the first part of this chapter we want to introduce the most important semi-structured data model on the Web, XML . Then we will, in the following parts study data management methods for these new data models, and then take a more abstract view by studying graph-oriented semi-structured data models, their close relationships with finite automata and applications of automata theory for indexing and structure extraction. Finally we will investigate their use for semantically annotating and integrating information sources on the Web.

# Today's Questions

1. What is XML ?
2. XML Syntax
3. XPath and XQuery

# 1. What is XML ?

## Data on the Web: HTML

- The Web is
  - A huge collection of hypermedia documents
  - A gateway to databases and applications
  - A platform for interacting with services (Web Services)
- It started with HTML
  - Markup language to define document layout
  - Hyperlinks for navigation
  - Interactive forms



The Web started as being a user interface to documents, databases and later interactive services. HTML was used as surface language to program the user interface

## Data on the Web: XML

- Limitations of HTML
  - Structure of data expressed as layout (example)
  - Semantics of data hard to analyse and difficult to share
  - No schemas, no constraints
- Thus XML (eXtensible Markup Language) has been developed
  - Markup language to define structured documents
  - Document schemas to fix the structure of documents
  - User-defined markup to express semantics
  - XML architecture for processing and extended functionality



Eventually data too was represented in HTML (tables, lists, matrices). It quickly turned out that this was not a good idea as it became hard to interpret the data correctly, not only for humans but even more for computers. The problem was that this lacked the good old wisdom that you should define a schema according to which your data is structured, before you populate a database. Therefore XML was invented. It transferred the merits of schemas into the document world. Document-orieneted people also like to say that they allow « user-defined » markup, as opposed to the « system markup » of HTML.

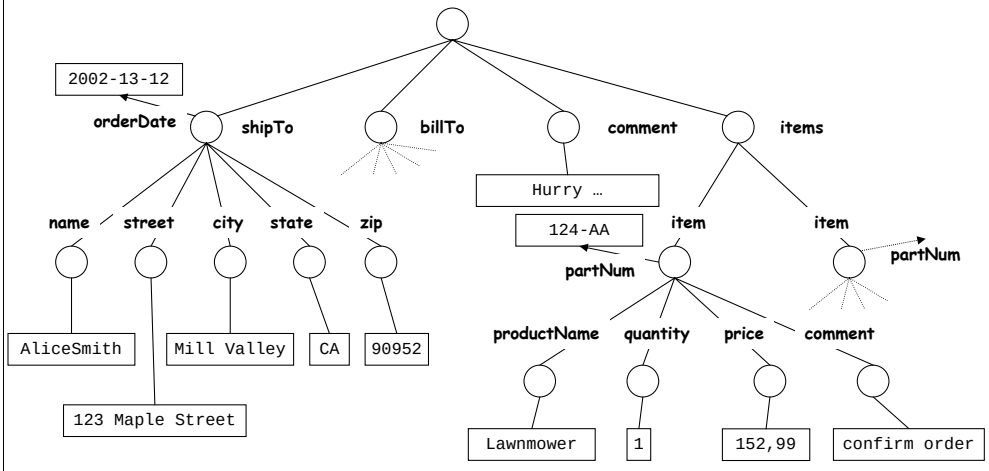
## Example: XML – as document language

```
<?xml version="1.0"?>
<purchaseOrder orderDate="1999-10-20">
  <shipTo country="US">
    <name>Alice Smith</name>
    <street>123 Maple Street</street>
    <city>Mill Valley</city>
    <state>CA</state>
    <zip>90952</zip>
  </shipTo>
  <billTo country="US">
    <name>Robert Smith</name>
    <street>8 Oak Avenue</street>
    <city>Old Town</city>
    <state>PA</state>
    <zip>95819</zip>
  </billTo>
  <comment>Hurry, my lawn is going wild!
  </comment>
  <items>
    <item partNum="872-AA">
      <productName>Lawnmower
      </productName>
      <quantity>1</quantity>
      <price>148.95</price>
      <comment>Confirm this is electric
      </comment>
    </item>
    <item partNum="926-AA">
      <productName>BabyMonitor
      </productName>
      <quantity>1</quantity>
      <price>39.98</price>
      <shipDate>1999-05-21</shipDate>
    </item>
  </items>
</purchaseOrder>
```



This is an example of an XML document. The most important thing to observe here is that it consists of a mix of markup, enclosed in <…> and textual content (everything between). The second observation is that the markup is hierarchically structured (indicated by the indentation). The syntactic details will be introduced later.

# Example: XML- as data model



# Data and Documents

- Serialization

**Document = medium for exchange of information**

**Communication**

**Information system 1**

**Information system 2**

## XML Document Type Definition

```
<!DOCTYPE purchaseOrder [
<!ELEMENT purchaseOrder (journals,conferences,
          books)>
<!ATTLIST purchaseOrder orderDate CDATA #REQUIRED>

<!ELEMENT shipTo (address)>
<!ATTLIST shipTo country CDATA #REQUIRED>
<!ELEMENT billTo (address)>
<!ATTLIST billTo country CDATA #REQUIRED>
<!ELEMENT address (name, street, city, state, zip)>

<!ELEMENT items (item*)>
<!ELEMENT item (productName, quantity, price, comment?)>
<!ATTLIST item partNum CDATA #REQUIRED>

<!ELEMENT comment (#PCDATA)>
<!ELEMENT name(#PCDATA)>
…
<!ELEMENT price(#PCDATA)>
]>
```



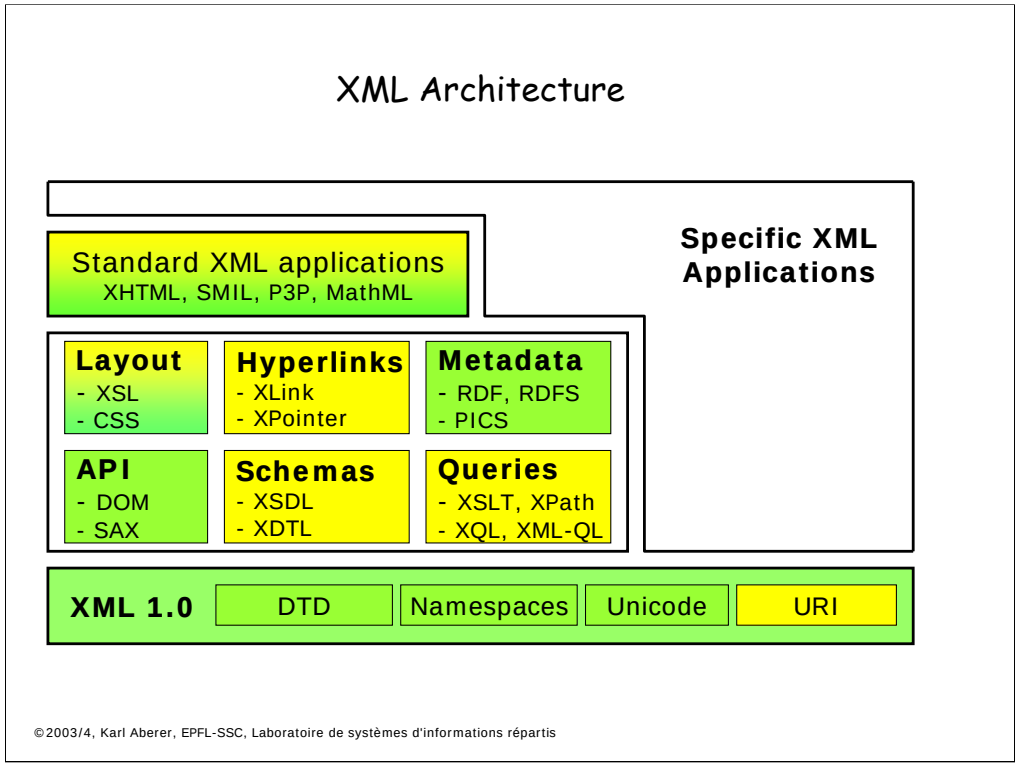
In addition, XML has a kind of schema language, XML Document Type Definitions (DTDs). They specify WHICH element tags may be used and HOW they can be used.

## What is XML ?

- Interpretation depends on the viewpoint and intended use
  - a language to describe the structure of documents.
  - foundation of the W3C architecture for Hypermedia documents on the Web.
  - the successor of HTML.
  - a method to represent structured data within text documents.
  - a standard data exchange format.
  - a data model for semi-structured (partially structured) data
- XML and Distributed Information Systems
  - XML as data model for managing semi-structured data
  - XML as canonical model to integrate heterogeneous data
  - XML as canonical data format to exchange data among information systems
- Tim Bray:
  "XML will be the ASCII of the Web – basic, essential, unexciting"



The interpretation of what XML is depends also on the viewpoint one takes: in the simplest case it may be considered as a document syntax, but depending on the context other interpretations are possible. It is a language to describe hierarchically structured text documents (this is also where the origin of XML lies, there was an earlier very similar language SGML from which XML was derived, and that was used by the publishing industry). By the W3C XML was chosen to be the foundation of the whole architecture of the Web, which explains also why it is called successor of HTML. For database persons XML is another way of how to structure data, with the additional advantage of getting a textual representation (or a serialization as it is often called) for free. The textual form of a XML document can thus also directly be used to exchange data via messages, which is why the EDI (Electronic data Interchange) community views XML nowadays as their standard message syntax. From the data perspective XML also lead to a more fundamental change in the way data management is perceived, namely the focus on the new question of how to manage, of which we have no or only partial knowledge of how it is structured. We will dedicate part 3 of this chapter mostly to answer this question.




The importance of XML lies not only in its inherent properties as document markup language, but also in the fact that it is used as the basis for a whole architecture covering almost every aspect of information processing imaginable. This architecture is developed under the auspices of the W3C consortium and builds on XML 1.0 which has been approved by the W3C. The architecture comprises the following components:

Programming API's: DOM and SAX, these are required to build applications that can process XML documents that have been parsed and made available in main memory.

Layout and hyperlinks: standardize the language constructs needed in order to represent the hypertext and layout properties of documents.

Programming APIs, Layout and Hyperlinks are covered in the lecture « Conception of Information Systems »

XML Data Management: XML Schema and Query provide the languages for managing large data-oriented documents and document-collections.

Metadata: as XML data will be exchanged and processed in many different places, it will often be necessary to provide additional data (information) about the data (metadata) in order to enable applications and users to correctly interpret and use the data.

We will learn in this lecture in detail about data and metadata management.

## Key XML Applications

- Web information systems
  - separation of layout and structure
  - better support to keep data consistent
  - reuse of data
  - more semantics available for more intelligent processing (personalization, agents, search engines)
- Electronic business
  - Integration of businesses processes
  - Standardization
  - messages and contracts represented in XML
  - replaces former EDI formats



As on the web information exchange is the basic activity, it is natural to use an exchange-oriented representation of information in the first place, and adapt the processing of the information, i.e. the data model, to the format in which the information is exchanged. Of course, this is a more complicated task, as XML was primarily not designed to serve as a data model for computer-internal processing. The difficulties and solutions to this extent will be part of this Chapter.

From an application perspective XML plays its role in two main areas:

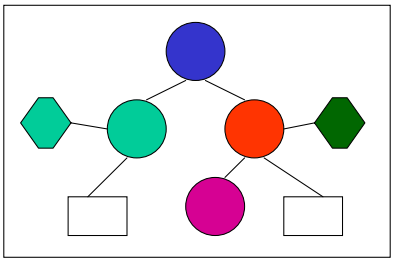
1. Information systems on the web: there the important progress with respect to HTML is the separation of layout issues from data structure. This allows to keep data consistent, reuse it, and to perform more intelligent processing of the data. This aspect of using XML is also the motivation for many issues that we learn in this course.
2. In electronic business XML plays its role as message format in implementing distributed processes within and across enterprises. It replaces earlier formats (EDI) and by being an authoritive standard it resolves some issues of heterogeneity (mainly at the level of syntax). This aspect is considered in the lecture « Conception of Information systems »

## Summary

- What is XML ?
- What is the difference between XML and HTML ?
- What is the difference between the XML syntax, architecture and applications ?
- What is the difference between data and document ?
- What are applications of XML ?

## 2. XML Syntax

- Well-formed XML conforms to a basic XML syntax and some semantic constraints for well-formedness
- Main concepts
  - Elements: used to structure the document, identify a portion of the document
  - Attributes: associate data values with elements, used to reduce the number of elements and for typed data
  - Character data (PCDATA): textual content of the document





```
<journal>
   <issue page=1>
      PCDATA,
      the content
   </issue>
   <issue page=2>
      <last/>
      more PCDATA
   </issue>
</journal>
```

Though XML supports schemas (by means of DTDs) XML documents are not required to have ones. Rather any document that following some basic syntax concerning the tags are valid XML documents. These are called well-formed XML documents. The constituents of an XML documents are elements (the tags), attributes and textual content. The tags must be nested, such that the elements form a hierarchy. Therefore it is always possible to view an XML document also as a tree, as illustrated by the example. It is important to consider always both views on XML, either as document – as in the ASCII (actually UNICODE) representation on the right -, or as data – as in the tree representation on the left (trees are a kind of data structure!)

## Well-Formed XML Syntax

- Syntax (excerpt)

```
document        ::=    prolog element Misc*
element         ::=    EmptyElemTag | STag content Etag
STag            ::=    '<' Name (S Attribute)* S? '>'
ETag            ::=    '</' Name S? '>'
Attribute       ::=    Name Eq AttValue
```

- Syntactic Properties (enforced by grammar)
  - single root element
  - tag names start with letter
  - tags must be properly nested
  - hierarchic structure induced by parents-child relationship
  - special syntax for empty tags

- Semantic Constraints
  - Start and end tag name must match
  - Attribute names within an element are unique



Well-formed XML can be specified by means of a formal syntax of which we illustrate only some important production rules. In particular one can see the hierarchical buildup for the production of element. This syntax implies some properties that well-formed XML documents have. However, there exist more requirements on well-formed XML, which are outside the scope of the syntax.

## XML Document Type Definitions (DTDs)

- Definitions
  - Definition of element and attribute names
- Content Model (regular expressions)
  - Association of attributes with elements
  - Association of elements with other elements (containment)
  - Order and cardinality constraints



XML document type definitions (DTDs) are used to specify

-which elements and attributes are allowed

-And how they can appear in the document in relation to each other

However, in difference to database schemas they are not given as types (a relational schema is for example a (data) type definition) but rather as a special kind of grammar.

## Element Declarations

- Basic form
  - `<!ELEMENT elementname (contentmodel)>`
  - `contentmodel` determines which other elements can be contained
  - Given by a regular expression
- Atomic contents
  - Element content
    `<!ELEMENT example ( a )>`
  - Text content
    `<!ELEMENT example (#PCDATA)>`
  - Empty Element
    `<!ELEMENT example EMPTY>`
  - Arbitrary content
    `<!ELEMENT example ANY>`



The main construct in DTDs is the declaration of elements. It consists of two parts, the name of the element and the content model, which is a regular expression that determines which other elements are allowed to appear within (or below) the element, and in which order and multiplicity. The content model is a regular expression built up from other element names. The atomic contents are other elements, text, which is represented by special built-in element name #PCDATA (=parseable character data), the empty content or any content, which imposes no further constraints.

## Element Declarations

- Sequence
  ```
  <!ELEMENT example ( a, b )>
  ```
- Alternative
  ```
  <!ELEMENT example ( a | b )>
  ```
- Optional (zero or one)
  ```
  <!ELEMENT example ( a )?>
  ```
- Optional and repeatable (zero or more)
  ```
  <!ELEMENT example ( a )*>
  ```
- Required and repeatable (one or more)
  ```
  <!ELEMENT example ( a )+>
  ```
  Mixed content
  ```
  <!ELEMENT example (#PCDATA | a)*>
  ```

- Content model can be grouped by paranthesis (nested expressions)
- Cyclic element containment is allowed in DTD (not document)



From the atomic content model one can construct composite content models, by using the standard regular expression operators sequence, alternative optional and repeatable. If text content occurs together with user-defined elements in the content model, this is called mixed content. The regular expression operators can be nested using parantheses and cyclic element containment is allowed. This allows for example to specify DTDs that allow documents of arbitrary depth.

## Attribute Declarations

- Each element can be associated with an arbitrary number of attributes
- Basic form
  - `<!ATTLIST Elementname  Attributename Type Default`
    `Attributename Type Default`
    `... >`

- Example:
  Document Type Definition

```
<!ELEMENT shipTo (  #PCDATA)>
<!ATTLIST shipTo    country CDATA #REQUIRED "US"
                    state CDATA #IMPLIED
                    version CDATA #FIXED "1.0"
                    payment (cash|creditCard) "cash">
```

  Document

```
     <shipTo        country="Switzerland"
                    version="1.0"
                    payment="creditCard"> … </shipTo>
```



Attributes are used to associate additional data with elements that are not represented as contents. Attributes are a heritage from the document processing world, where the elements have been used to structure the documents, and attributes were used to specify instructions for document processing. From a data modeling viewpoint, in many cases attributes and elements can be used interchangeably, and the preference is a matter of taste and capabilities of the XML processing environment.

## Attribute Declarations - Types

- CDATA
  - String
  - `<!ATTLIST example HREF CDATA #REQUIRED>`
- Enumeration
  - Token from given set of values, Default possible
  - `<!ATTLIST example selection ( yes | no | maybe ) "yes">`
- Possible Defaults
  - Required attribute: `#REQUIRED`
  - Optional attribute: `#IMPLIED`
  - Fixed attribute: `#FIXED "value"`
  - Default for enumeration: `"value"`
- Other attribute types: IF, IDREF, ENTITY, ENTITIES, NOTATION, NAME, NAMES, NMTOKEN, NMTOKENS



Attributes can be typed. The standard type of an attribute is CDATA, i.e. Strings. Enumerations allow to specify finite sets of data values (note that the same could be achieved by defining an appropriate DTD, with an empty element type for each data value.) The ID/IDREF mechanisms is one case where attributes' expressive power goes beyond that of elements. It allows to specify references WITHIN documents. It is required in the XML specification that an XML parser must check referential integrity of those references. A number of other attribute types witness the origins of XML in the document processing world, and are not of relevance for our further use of XML. The most notable among those are entities, which provide a kind of macro mechanism, that allows to factor out repeating parts in the documents and document type definitions.

# ID/IDREF

- ID, IDREF
  - ID is a unique identifier **within** the document
  - IDREF is a reference to an ID
  - Referential integrity checked by the parser
  - ID's determined by the application
  - 
```
<!ATTLIST example       identity   ID    #IMPLIED
                        reference  IDREF #IMPLIED>
```

# Example: ID/IDREF

**DTD fragment:**

```
<!ATTLIST fig          id      ID     #IMPLIED>
<!ATTLIST figref       refid   IDREF  #IMPLIED>
```

**Document fragment:**

```
<chapter>
  <title>Apples<\title>
  <para>
    <fig id="1">
     <caption> this is a figure<\caption>
    <\fig>
  <para>
<\chapter>
<chapter>
  <title>Frogs<\title>
    <references>
      <figref refid="1"\>
    <\references>
<\chapter>
```

## Usage of DTDs

**External DTD Declaration**

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE test PUBLIC "-//Test AG//DTD test V1.0//EN"
               SYSTEM "http://www.test.org/test.dtd">
<test> "test" is a document element </test>
```

**Internal DTD Declaration**

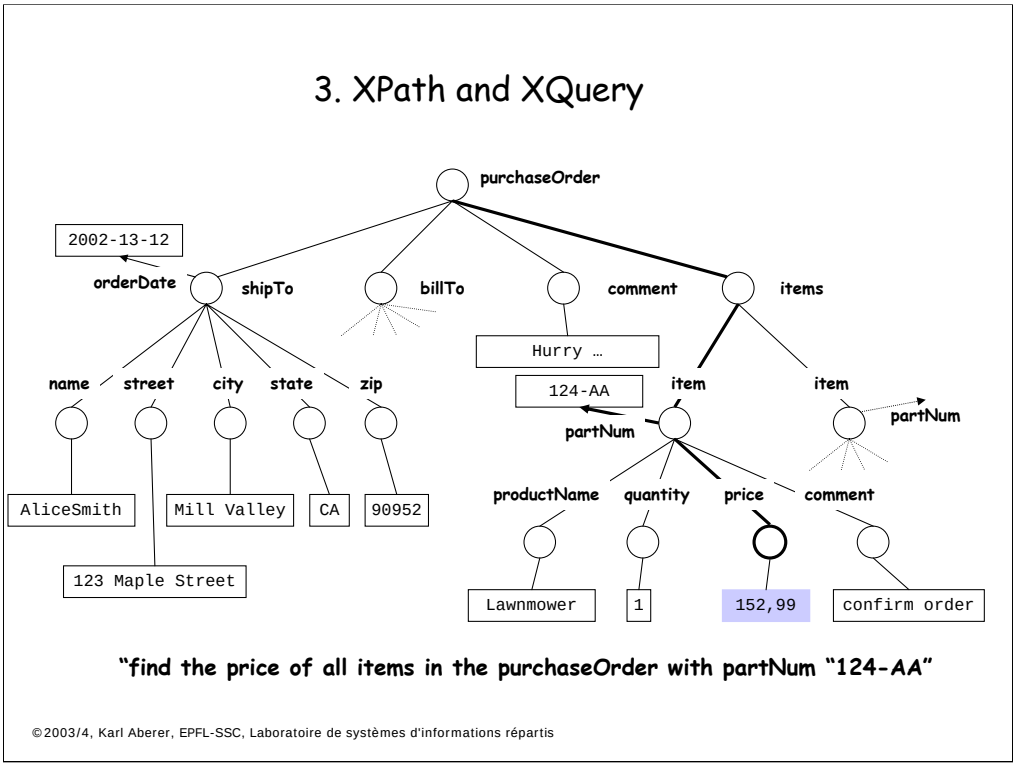
```
<!DOCTYPE test [ <!ELEMENT test EMPTY> ]>
<test/>
```



The DTD can be stored in a separate document, which is useful when it is shared by many applications/documents, or be directly enclosed into the document, which is useful when a document is exchanged and the DTD is not known by the receiver. It is also possible to include only parts of the DTD in the document, which makes practical sense only when using the entity mechanism.

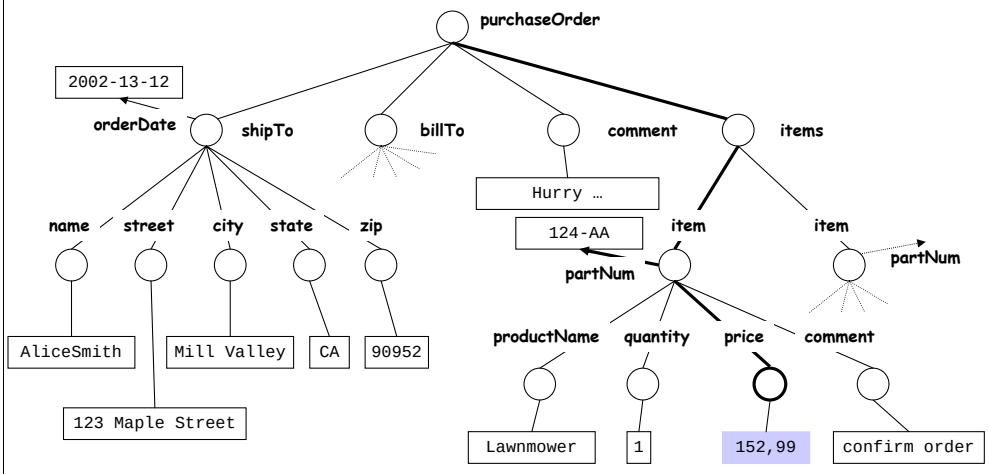
## Summary

- Which syntactic and semantic constraints are imposed on a well-formed XML document ?
- Which atomic element types and operators can be used to build an element content model ?
- Which default values exist for attributes ?
- What is the ID/IDREF mechanism ?
- Is a DTD always a separate document ?




The problem to address is the following: given an XML document, how can we answer queries as the following: “find the price of all items in the purchaseOrder with partNum “124-AA”. For relational databases this would be a typical query to be expressed in SQL. We introduce now the corresponding counterparts for XML.

# Example XPath Query




**/purchaseOrder/items/item[@partnum="124-AA]/price**

## XPath Overview

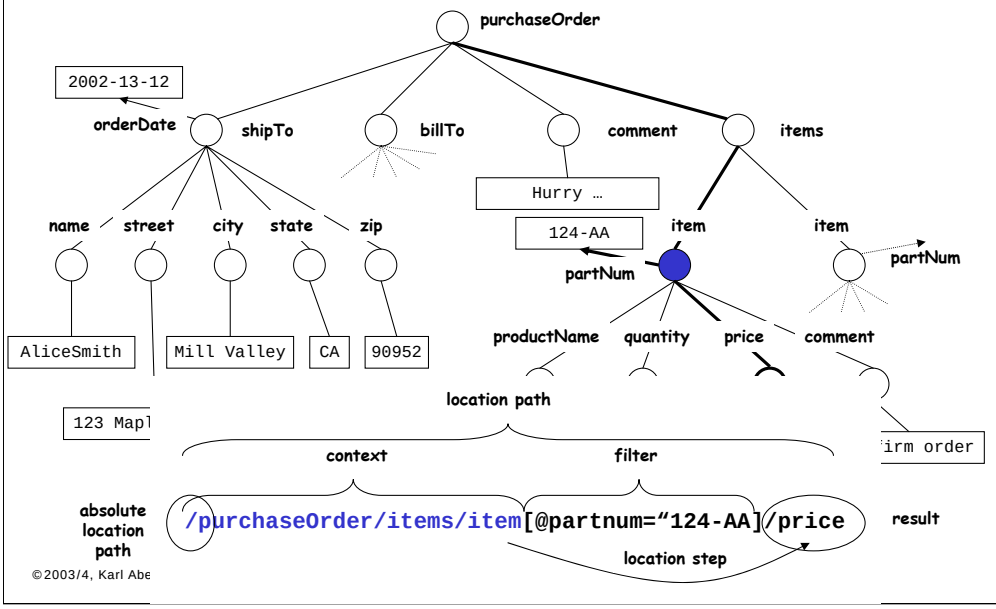
- Logical adressing of document parts
- Key concepts
  - Location paths select nodes relative to a given context node
  - Absolute location paths start at the document root
  - Location paths consist of a sequence of location steps
  - The last location step determines the result
  - Filter expressions contain location paths
  - Context of filter expressions is the associated location step
- Practical aspects
  - Non-XML Syntax
  - Used by other XML standards (XSLT, Xpointer)
  - Used within XML attributes and URIs



When considering XML documents as data, we need to provide a query language to access this data. As the data view is a tree representation, a first capability of any XML query language is to navigate along the paths in the tree. This is the essential function that Xpath provides. It allows to address (specify) document parts by providing a navigation « instruction » how the part can be reached. The basic principle of Xpath is to navigate in tree document in a manner comparable to the navigation in a directory tree in a file system and to evaluate at each step additional filter conditions. The navigation steps (location steps) and the evaluation of filter conditions depend on the current navigation context (the place in the tree where the navigation has arrived). It is important to understand that the result of an Xpath query is always a set of element (or attribute) nodes (and nothing else, in particular not a XML document fragment)

From a practical viewpoint it is interesting to mention that Xpath does not use an XML syntax, thus an Xpath expression is not a well-formed XML document (other standards, e.g. XSLT, the XML document transformation language, use XML syntax to denote expressions in their specific model). Xpath is a component that is reused in other standards, notably in XSLT. Xpath expressions are also intended to extend URLs to URIs (universal resource identifier) to address parts of XML documents.

# Example XPath Query



## XPath Location Paths

- A location step consists of
  - an axis (the navigation direction),
  - a node test, and
  - a predicate
- Axis operators
  - `AxisName ::= 'ancestor' | 'ancestor-or-self' | 'attribute' | 'child' | 'descendant' | 'descendant-or-self' | 'following'| 'following-sibling' | 'namespace' | 'parent' | 'preceding' | 'preceding-sibling' | 'self'`
- Example: absolute location path

```
/child::purchaseorder[child::shipto/child::name="Alice"]/
     child::items/child::item[position()=1]
```

- Abbreviated syntax (used in practice)

```
/purchaseorder[shipto/name="Alice"]/items/item[1]
```



The simple navigation pattern in the previous example (downwards to elements with specific names) is generalized in Xpath. Each navigation step is characterized by three things:

The direction, i.e. a navigation needs not necessarily move from a parent to a child node, but can follow any relation among elements, in particular by traversal of elements at the same tree level according to the document order (following, preceding) and traversal of multiple nodes (descendants). A complete account of the possible navigation operators is given. The node test checks whether a element node that is encountered in the navigation matches a certain element name. And finally the predicates are the filter expressions which allow to select nodes based on other properties than their name. In particular, filters allow to use other Xpath queries to check a property of an element. In that case the predicate is considered as successfully evaluated if a non-empty result set is generated. In order to take account of the different axis operators an extended syntax is used in Xpath that specifies for each location step the axis operator. In practice however, the abbreviated syntax that we have already seen is more common.

## XPath Abbreviated Syntax

- Selection of elements
  - `item`
    - selects from the current context all elements with name `item`
- Hierarchy operators
  - `item/price`
    - all `price` elements within `item`
  - `./item`
    - equivalent to `item`
  - `purchaseOrder//item`
    - all descending elements with name `item`
  - `name/..`
    - parent node of `name`



We introduce all operators of the abbreviated syntax by means of examples. The hierarchy operators are straightforward to understand as their semantics coincides with the UNIX directory navigation semantics, except the // operator. It represents the navigation to all descendent elements of the current context element, i.e. all elements on the paths to the leafs are selected (and not only the leaf nodes).

## XPath Abbreviated Syntax

- Wildcards
  - `purchaseOrder/*/item`
    - all `item` that are reachable by passing through one arbitrary element
  - `*/*`
    - all grandchildren
  - `@*`
    - all attributes
- Indexed Access
  - `item[1]`
    - first `item` element
  - `item[1, 4]`
    - first and fourth `item` element
  - `item[last()]`
    - last `item` element



Wildcards match all element names, in other words no node test is performed. Be careful: the wildcard * has in Xpath another meaning than in a regular expression or in an XML DTD, where it indicates repeated occurrence, which is covered in turn in Xpath by the // operator. Also attributes can be selected. This is particularly useful in filter expressions when attribute values need to be checked.

Indexed access allows to traverse neighbouring elements at the same tree level. We see here also one example of using a (built-in) function in a predicate, namely last(). A number of basic functions for node access and string manipulation are specified in Xpath.

## XPath Abbreviated Syntax

- Filter
  - `item[price]`
    - all `item` elements containing a `price` element
  - `purchaseOrder[billTo]/items[item]`
    - All `items` elements containing an `item` element that are contained in a `purchaseOrder` element containing a `billTo` element
  - `purchaseOrder[items/item]`
  - `purchaseOrder[shipTo and billTo]`
  - `item[productName=« car »]`
    - All `item` elements containing a `productName` element with textual content « car »
  - `item[@partNumber=« A100 »]`

- Union
  - `purchaseOrder/billTo/name | purchaseOrder/shipTo/name`
  - Only at top level !



We see that two types of filter expressions exist. One that returns an element (or attribute) set. These are true if the sets are non-empty. In this way also Boolean combinations of Xpath expressions (e.g. shipTo and billTo) make sense. And others returning a Boolean value (e.g. using the equality predicate). These are true if for at least one element in the Xpath expression appearing in the predicate the condition is satisfied. Xpath provides only one set operator for set union (in contrast to SQL) and this with the restriction that it can only be applied at the top level of the Xpath expression.

## XPath Abbreviated Syntax Operators

| | | |
|---|---|---|
| / | Child operator | document element that is the direct child |
| // | Recursive descent | all elements in the document that are indirect child |
| . | Current context node | |
| * | Wildcard | matches all element and attribute names |
| @ | Attribute | distinguishes attributes from elements |
| [] | Filter | |
| f() | Method call | |
| () | Grouping | |

## XQuery - Querying XML Data

- Problem: XPath lacks basic capabilities of database query languages, in particular join capability and creation of new XML structures
- XQuery extends XPath to remedy this problem
- Additional concepts in XQuery
  - Extended path expressions
  - Element constructors
  - FLWR expressions
  - Expressions involving operators and functions
  - Conditional expressions
  - Quantified expressions



By now it should have become clear that Xpath lacks basic capabilities one would expect from a (declarative, set-oriented) database querying language. In particular it has no general support for set operators, it allows to return only element and attribute sets and is thus not closed and it has no support of an operation that is equivalent to a relational join, which would be required to establish value-based relationships among XML document parts. We introduce now the most important additional concepts of XQuery as they were specified in June 2001. The goal is not to obtain a thorough knowledge and capability to use Xquery, but to understand the substantial additional concepts to Xpath and the relationships to SQL. We must be aware that this standard is not yet finalized and we may expect certain changes (probably not major ones).

With a knowledge of Xpath and SQL the following presentation of Xquery should be straightforward to follow.

## Dereference Operator

- `document("zoo.xml")//`
  `chapter[title = "Frogs"]//figref/@refid->fig/caption`
  - Find captions of figures that are referenced by `figref` elements in the chapter of "`zoo.xml`" with title "`Frogs`".

```
document.xml:
<chapter>
  <title>Apples<\title>
  <para>
    <fig id=« 1 »>
     <caption> this is a figure<\caption>
    <\fig>
  <para>
<\chapter>
<chapter>
  <title>Frogs<\title>
    <references>
      <figref refid=« 1 »\>
    <\references>
<\chapter>
```



Navigation in Xquery is possible over IDREF attributes. In this example first the figref element would be located from which the IDREF value is taken and the fig element with that ID value is located, leading to the result.

The expression document(« zoo.xml ») replaces the / operator.

## Element Constructor and FLWR Expressions

```
<result>
{
   LET $a := avg(document("bib.xml")//book/price)
   FOR $b IN document("bib.xml")//book
   WHERE $b/price > $a
   RETURN
       <expensive_book>
          {$b/title}
          <price_difference>
             {$b/price - $a}
          </price_difference>
       </expensive_book>
}
</result>
```

"macro"

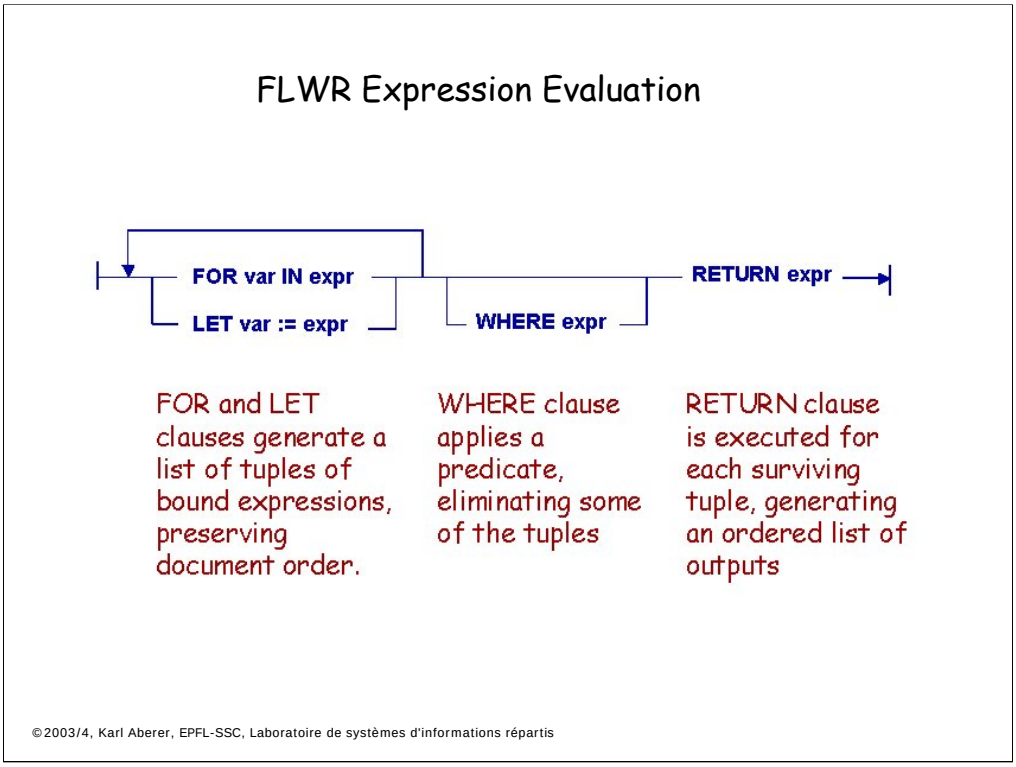
"FROM"

"SELECT"

- For each book whose price is greater than the average price, return the title of the book and the amount by which the book's price exceeds the average price



This query exhibits a whole wealth of new concepts of Xquery. Most notably one can see that Xquery allows variable binding as SQL. Different to SQL where relations are available to bind variables in Xquery they have to bind to sets that result from other Xquery expressions (this is the only possibility to obtain sets). This is done in the FOR clause. In addition using the LET clause one can introduce variables that factor out repeatedly occurring expressions in the queries. Note that these variables are used very differently from the ones bound to set valued expressions: they are just syntactically replaced in the query. The second observation is that new a WHERE clause is available to express conditions. This allows in particular to express joins when multiple variables are bound in the FOR clause. The third observation is that a RETURN clause allows to return structured results, creating new XML document fragments. Finally we see that a query expression itself can be nested within a XML document fragment.




The semantics of Xquery expressions is defined similarly to SQL (which is in short: build the Cartesian product of the relations in the FROM clause, evaluate the predicates in the WHERE clause and then project on the attributes in the SELECT clause). Also for FLWR expression first generate all tuples from the Cartesian product space of all sets to which variables are bound. An important difference that the order among document elements needs to be preserved, therefore also the order in which the variables appear in the FOR clause has an impact on the order the result tuples will be sorted. The WHERE clause is evaluated as for SQL and for each remaining tuple an XML document fragment is generated by replacing the variables by the tuple values. There is also a XML query algebra under development which is intended to provide a precise semantics to Xquery. As all these are ongoing developments we will not further delve into the technical details, as the final outcome of these developments is not yet determined.

# Join Example

```
<result>
{
FOR $seller IN document("users.xml")//user_tuple,
     $buyer IN document("users.xml")//user_tuple,
     $item IN document("items.xml")//item_tuple,
     $highbid IN document("bids.xml")//bid_tuple
WHERE
     $seller/name = "Tom Jones" AND
     $seller/userid = $item/offered_by AND
     contains($item/description, "Bicycle") AND
     $item/itemno = $highbid/itemno AND
     $highbid/userid = $buyer/userid AND
     $highbid/bid = max(document("bids.xml")//bid_tuple
            [itemno = $item/itemno]/bid)
RETURN
     <jones_bike>
     { $item/itemno } { $item/description }
            <high_bid> { $highbid/bid } </high_bid>
            <high_bidder> { $buyer/name } </high_bidder>
     </jones_bike>
SORTBY(itemno) }
</result>
```

assumes an XML representation of a relational database

## Functions

- NAMESPACE
  xsd=http://www.w3.org/2001/03/XMLSchema-datatypes

```
FUNCTION depth(ELEMENT $e) RETURNS xsd:integer
{   -- An empty element has depth 1
    -- Otherwise, add 1 to max depth of children
  IF empty($e/*) THEN 1
  ELSE max(depth($e/*)) + 1 }

depth(document("partlist.xml"))
```

  - Find the maximum depth of the document named "partlist.xml."



Xquery goes much further in terms of expressiveness than SQL by allowing the definition of arbitrary user-defined functions. This is a feature that can be found in SQL99 (as well), the object-relational query language standard building on SQL92. We also see in this example, that Xquery uses type specifications that are provided in the XML Schema standard, which too is currently under development.

## Existential and Universal Quantifiers

- ```
  FOR $b IN //book
  WHERE
     SOME $p IN $b//para SATISFIES contains($p, "sailing")
     AND contains($p, "windsurfing")
  RETURN $b/title
  ```
  - Find titles of books in which both sailing and windsurfing are mentioned in the same paragraph.
- ```
  FOR $b IN //book
  WHERE
    EVERY $p IN $b//para SATISFIES contains($p, "sailing")
  RETURN $b/title
  ```
  - Find titles of books in which sailing is mentioned in every paragraph.



As in SQL, Xquery also supports the concepts of universal and existential quantification of variables ranging over set expressions.

## XQuery Development

- Conciliates many approaches
  - Path expressions from XPath
  - Variable binding concept from XML-QL
  - SELECT-FROM-WHERE paradigm from SQL92
  - Orthogonality from OQL
  - User-defined functions from SQL99
- Semantics given in terms of the XML query algebra



We have seen that Xquery unifies concepts from many different origins. The variable binding and result generation concept used was in fact for the first time specified in a research system, XML-QL. Most of the other ideas are borrowed from the relational and object-relational query languages. In particular the orthogonality reminds a lot of OQL (Object Query Language): orthogonality means that every where in an expression where it is type-safe, an arbitrary expression of the language can be substituted. This is for example not true for SQL92.

We have to point out the Xquery includes many other technical details which we will not consider further as they are conceptually not very interesting.

## Summary

- What is the relationship between XPath and XQuery ?
- Which result types are possible in XPath ?
- What is an axis in XPath ?
- What is the difference between a/*/b and a//b ?
- How can a join operation be specified using XQuery ?
- What are differences and commonalities among XQuery and SQL92 ?

# References

- WebSite
  - XML, XPath, XQuery: http://www.w3.org/
  - XQuery Engine: `http://demo.openlinksw.com:8391/xquery/demo.vsp`
- Book
  - S. Abiteboul, P. Bunemann, D. Suciu: Data on the Web: From Relations to Semistructured Data and XML, Morgan Kaufman, 2000.