Chapter 5: XPath/XQuery Data Model

References:

- Mary Fernández, Ashok Malhotra, Jonathan Marsh, Marton Nagy, Norman Walsh (Ed.): XQuery 1.0 and XPath 2.0 Data Model (XDM). W3C Recommendation, 23 January 2007, `http://www.w3.org/TR/xpath-datamodel/`
- Ashok Malhotra, Jim Melton, Norman Walsh (Ed.): XQuery 1.0 and XPath 2.0 Functions and Operators. W3C Recommendation, 23 January 2007. `http://www.w3.org/TR/xpath-functions/`
- John Cowan, Richard Tobin (Editors): XML Information Set (Second Edition). W3C Recommendation, 4 February 2004, `http://www.w3.org/TR/xml-infoset`
- Jonathan Marsh (Ed.): XML Base. W3C Recommendation, 27 June 2001, `http://www.w3.org/TR/xmlbase/`
- G. Ken Holman: Definitive XSLT and XPath. Prentice Hall, 2002, ISBN 0-13-065196-6, 373 pages.
- Priscilla Walmsley: Definitive XML Schema. Prentice Hall, 2001, ISBN 0130655678, 560 pages.
- W3C Architecture Domain: XML Schema. `http://www.w3.org/XML/Schema`
- Paul V. Biron, Ashok Malhotra: XML Schema Part 2: Datatypes. W3C, 28. October 2004, Second Edition `http://www.w3.org/TR/xmlschema-2/`

Objectives

After completing this chapter, you should be able to:

- draw the XDM (XPath/XQuery Data Model) Tree representation for a given XML document.
- explain the most important XDM node types and their essential properties.
- define "document order".
- mention some details, in which XML data files with the same XDM tree might differ.

Overview

Introduction (1)

- XML documents can be understood as an external representation of a tree. This tree is the real information content of the document.

  Such as 3, 03 and +3 are all the same number, there are some variations possible in XML documents that are considered as unimportant and not represented in the tree, see below. [From the XPath Specification:] "XPath operates on the abstract, logical structure of an XML document, rather than its surface syntax."

- The tree is (usually) the result of parsing the XML document, and possibly validating it.

  The validation (wrt DTD/schema) might change the tree, e.g. add default values. The result is called "PSVI" (post-schema-validation infoset). The tree might also be directly constructed via API calls.

Introduction (2)

- Query and transformation languages (e.g. XPath, XQuery, XSLT) are defined in terms of such a tree.
- Also parser interfaces, such as DOM, have a tree structured view of the document.
- There are several standards:
  - ◇ XML Information Set (Infoset)
  - ◇ XQuery and XPath Data Model (XDM)
  - ◇ Document Object Model (DOM)

  Although all view XML documents as trees, they are different.

Introduction (3)

- The XML Information Set (Infoset) Recommentation states: "This specification provides a set of definitions for use in other specifications that need to refer to the information in an XML document."
- It only lays the foundation (a common reference framework) for other specifications.
- It talks about "information items" (with properties), not nodes or objects.
- This information could be made available to applications also through an event-based interface.

Introduction (4)

- The XDM specification states
  - ◇ how to construct an XDM instance from an infoset
  - ◇ how to construct an XDM instance from a PSVI (post-schema-validation infoset)
  - ◇ how to construct an infoset from an XDM instance.
- In this way, the XDM specification can use certain notions already defined in the infoset specification.

Introduction (5)

- There is not a one-to-one correspondence between XML infosets and instances of the XDM. E.g.:
  - ◇ XML Infoset permits trees that contain unparsed general entities, XDM assumes that they are fully expanded.
  - ◇ In XML Infoset (as in XML) the document node can have only one element child, XDM permits several element child nodes (and also text nodes as children, which are not in Infoset).

Introduction (6)

- Examples for differences Infoset vs. XDM, cont.:
  - ◇ XDM permits document fragments, sequences of document nodes, sequences of atomic values, and sequences mixing nodes and atomic values. All this is not in the Infoset standard.

    Necessary as intermediate values for functions.
  - ◇ In Infoset, attribute information items have a property “`specified`”, which permits to check whether an attribute value was actually specified or defaulted. In XDM, this is indistinguishable.

Introduction (7)

- Whereas the Infoset standard can be understood without knowing XML Schema, XDM requires at least some knowledge of it.

  Theoretically, it should be possible to understand XDM without knowing the details of XML syntax. After all, what is really important are the data structures, not how they are represented externally. However, the dependency on XML schema then becomes a problem.
- It would be possible that an XML DBMS uses the interface defined in the XDM standard as an API for working with query results.

  However, applications probably expect a DOM interface (which they would use when accessing XML data in a file via an XML parser).

Overview

Unessential Details (1)

- Not all syntactic details of an XML document are contained in the internal tree representation.
- For instance, there is no way to find out whether an attribute value was written with ' or ". Only the value itself is made available to the application.
- When a document is parsed, internally modified (e.g., by XSLT) and printed (“serialized”) again, this information is most probably not available.

  The output style might be configurable, but at least the same type of quotes will be used for all attributes (increases uniformity).

Unessential Details (2)

Syntactic Details that are not in the Information Set:

- Certain white space:
  - ◇ in tags (except in attribute values),

    Note that this means that all nice formatting of long tags split between multiple lines is lost.
  - ◇ outside the document element,
  - ◇ immediately following the target name of a processing instruction.
- Order of attributes within a start tag.
- Kind of quotation marks (' or ") used for attribute values.

Unessential Details (3)

Details that are not in the Infoset, continued:

- Kind of line delimiter used (LF, CR, CR-LF).
- The difference between the two forms of an empty element: `<E/>` vs. `<E></E>`.
- Whether a character is represented as a character reference.
- Boundaries of CDATA marked sections.
- Boundaries of general parsed entities.
- System and public ID of the document type.

Unessential Details (4)

Details that are not in the Infoset, continued:

- Most information from the DTD:
  - ◇ content models,
  - ◇ grouping and ordering of attribute declarations,
  - ◇ comments in the DTD,
  - ◇ order of declarations,
  - ◇ location of declarations (internal/external),
  - ◇ boundaries of conditional sections and parameter entities in the DTD,
  - ◇ ignored declarations (including redefinition).
  - ◇ default value of attributes (unless used).

Unessential Details (5)

Problem with `pattern`-Facet:

- In XDM, only the internal representation of data values is stored (i.e. the value itself, not the lexical representation).

  Thus, the differences between distinct lexical representations of the same value are considered unessential (e.g. 3, +3, 03).
- However, the `pattern`-facet refers to the lexical representation.
- It is not guaranteed that when a value is printed, a lexical representation is constructed that satisfies the `pattern`.

Unessential Details (6)

Problem with `QName`-Type:

- In XML schema,
  - ◇ the external representation of `QName` values consists of a prefix and a local name, whereas
  - ◇ the internal representation consists of a namespace URI and a local name.
- In XDM, such values are a triple consisting of
  - ◇ a prefix (possibly empty),
  - ◇ a namespace URI (possibly empty/absent),

    If the URI is empty, the prefix must be empty. (converse allowed.)
  - ◇ a local name.

Unessential Details (7)

Problem with `QName`-Type, continued:

- The reason for making the prefix part of the value is to simplify the printing of `QName`-values (i.e. the mapping from internal to external representation).
- For `QName`-values that are part of a document, a prefix could always be determined (but it might be not unique).
- However, the data model also permits atomic `QName` values that are not part of a document.

Unessential Details (8)

Problem with `QName`-Type, continued:

- Two values are (mostly?) treated as equal if they differ only in the prefix.

  This ensures that the new definition is compatible with the old one.
- The XDM standard lists numerous consistency rules to ensure that if a `QName`-value appears in a context that can have a namespace declaration, there really is one for this prefix-URI pair.

  Including a default namespace declaration if the prefix is empty, but the URI is not. Furthermore, `QName` values with prefix are forbidden in attribute nodes without parent.

Overview

Type System (1)

- XDM is based on the type system of XML Schema.
- However, a few new types were introduced in the type hierarchy of XML Schema.
- Some of these types are needed for documents (or parts of documents) that where not validated.

  Partial validation occurs in case of wildcards with skip or lax mode. In XDM, every value has a type.
- The new types are in the XML Schema namespace, they will probably be added in the next version.

Type System (2)

- The new types are (continued on next page):
  - `untyped`: For element nodes that have not been validated.
  - `untypedAtomic`: For attribute values that have not been validated, and text (e.g. element content) that has not been assigned a more specific type.
  - `anyAtomicType`: A base type for all atomic types.

    XML Schema only has `anySimpleType`, which also contains list and union types. XDM only contains atomic types (but permits sequences). `anySimpleType` is the base type of `anyAtomicType`.

Type System (3)

- New types, continued:
  - `dayTimeDuration`: Derived from `duration` with a pattern that permits only days, hours, minutes, seconds (including fractional seconds).

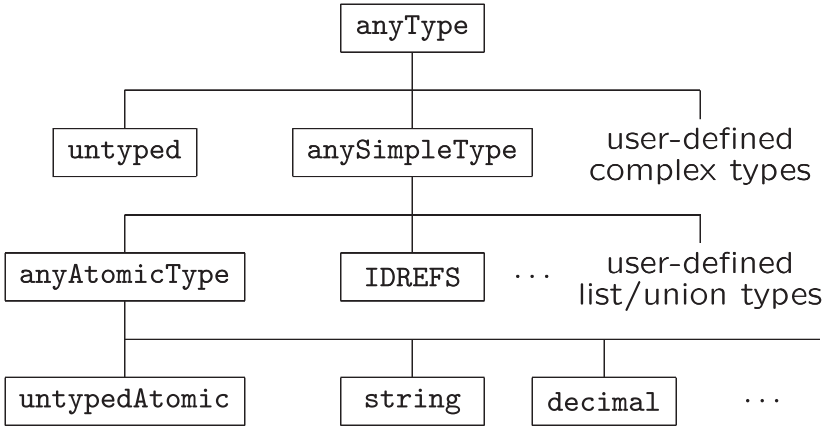
    In this way, it can be represented as the total number of seconds.
  - `yearMonthDuration`: Derived from `duration` by permitting only year and month components.

    In this way, it can be internally represented as the total number of months. It might be inconsistent with XML Schema that it does not have the prefix "g".

Type System (4)

- In addition, it seems that the XDM committee had a slightly different view on `dateTime` values than the XML Schema committee:
  - In the XDM data model, these values are represented as 7-tuples consisting of year, month, day, hour, minute, second, timezone.
  - XML Schema explains them with two time axis (one for UTM, the other for local time) measured in seconds. The specific timezone is lost.
  - XDM: "Leap seconds are not supported."

Type System (5)



Basic Definitions (1)

Atomic Types:

- An atomic type is a primitive simple type or a type derived by restriction from such a type.

  There are 23 primitive simple types: `string`, `boolean`, `decimal`, `float`, `double`, `duration`, `dateTime`, `time`, `date`, `gYearMonth`, `gYear`, `gMonthDay`, `gDay`, `gMonth`, `hexBinary`, `base64Binary`, `anyURI`, `QName`, `NOTATION`, `untyped`, `untypedAtomic`, `anyAtomicType`, `dayTimeDuration`, `yearMonthDuration`.

- Types derived by `union` or `list` are not atomic.

  XDM does not need values that have list or union types. Values of list type are represented as a sequence (see below), values of union type are assigned the more specific type the concrete value belongs to (or one of these types, if there are several). Union types are needed for static type declarations, but XDM is about concrete values.

Basic Definitions (2)

Type Identification:

- Types are identified by QNames (expanded by the prefix as explained above), e.g.

  `(xs, 'http://www.w3.org/2001/XMLSchema', integer)`

- This also holds for user-defined types: The namespace is the target namespace of the schema.

- For anonymous types, the processor must construct a unique name.

  This name is implementation-dependent.

Basic Definitions (3)

Atomic Value:

- An atomic value is a value in the value space of an atomic type and is labelled with the name of that type.

- The standard uses this notation in an example:

  `xs:anyURI("http://www.example.com/catalog.xml")`

  This basically looks like a call to a constructor function for the type. It is understood that `xs` is bound to `http://www.w3.org/2001/XML-Schema`.

- Working with pairs of type ID and binary value is common in dynamically typed ("untyped") programming languages (similar to "variant record").

Basic Definitions (4)

Node:

- A node is an object with a unique identity and properties.

  I.e. there can be distinct nodes that agree in all properties. This unique identity is intrinsic to the data model, it is different from the unique identity assigned by the user to some nodes with an attribute of type `ID`.

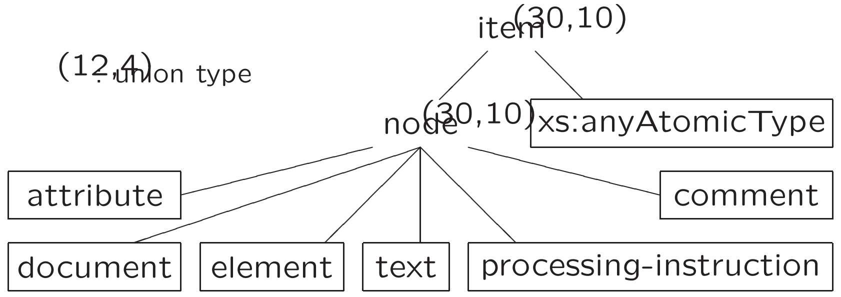
- There are seven kinds of nodes (subclasses): document, element, attribute, text, processing instruction and (possibly) namespace nodes.
- The node properties are explained below.

Basic Definitions (5)

Item:

- An item is either a node or an atomic value.

Type System, continued:



Basic Definitions (6)

Sequence:

- A sequence is an ordered collection of zero or more items (a list). It may contain duplicates.

  A sequence may contain a mixture of nodes and atomic values. The same node may be contained in several sequences.

- Every instance of the data model is a sequence.

  An instance of a data model (e.g., the relational model) is usually a database state. However, an XML database might store several (or even many) documents (when one parses a document, one gets an XDM instance). Furthermore, data values returned from functions are also instances of the XDM.

Basic Definitions (7)

Sequences, continued:

- Sequences replace the node-sets in XPath 1.0.
  - ◇ Node-sets could not contain duplicates, whereas sequences can.
  - ◇ One must now explicitly use a function for duplicate removal.

    Also in SQL, `DISTINCT` must be used explicitly.

  - ◇ The elements of a node set had a fixed order (“document order”, see below), in sequences any order is possible.

Basic Definitions (8)

Sequences, continued:

- Sequences cannot contain sequences.
  - ◇ If a sequence is inserted into another sequence, it is automatically "unnested".

    The structure is "flattened".
- Sequences of length one ("singleton sequences") and items (atomic values or nodes) are identified.
- Empty sequences are used like a null value.

  (e.g., for non-applicable properties).

Overview

1. Introduction
2. Internal vs. External Representation
3. Basic Definitions (Types, Sequences, . . .)
4. Node Types, Example(160,16)

Nodes: Subclasses

XDM has seven Kinds of Nodes:

- document
- element
- attribute
- text
- namespace (may be left out in implementation)
- processing instruction
- comment

Nodes: Properties (1)

- Nodes have properties. Each kind of node has a different set of properties. In addition, the standard defines accessor functions for these properties.
- In most cases, the correspondence is 1:1, and property and accessor function have the same name. But there are exceptions.

  Was this confusing duplication was really necessary? Of course, it is standard in object-oriented programming to have stored values in the objects (the properties) and accessor functions. However, at least in the case of the properties "`string-value`" and "`typed-value`" the standard explains that it might not be necessary to store both. Also the `string-value` of element nodes (a property) can be computed from the string values of its descendants (so it should not be stored).

Nodes: Properties (2)

- The XDM standard defines 17 such accessor functions, but it is not required that these are really made available to the user as functions on the nodes.

  Currently, an implementation of XDM is always part of a larger implementation (e.g., an XSLT or XQuery implementation). Therefore, it is not necessary to prescribe the internal interface used to access nodes. These functions are only a proposal or an illustration of the information that should somehow be made available.

- For instance, in XPath, only some of the properties can actually be used as functions, some are implicitly used by the path expressions.

Nodes: Properties (3)

Example:

- XDM defines an accessor function "`node-kind`" that returns the kind of the node as string.

  Internally in an implementation, it might be more appropiate to use an enumeration value than a string.

- There is no corresponding property.

  The accessor functions can be seen as pure virtual functions declared in the abstract superclass "`node`" and implemented in each subclass. Then it is easy to return a constant value in each implementation.

- There is no XPath function with this name.

  However, most "axis" return only nodes of a single kind, otherwise the "node test" can be used to check for a specific kind of node.

Nodes: Properties (4)

- All accessor functions are defined on all seven kinds of nodes (whereas the properties are specific to each kind of node).

  This corresponds to the view that the accessor functions are declared in the abstract superclass "`node`".

- If an accessor function corresponds to a property that is not applicable to the current node, it returns the empty sequence.

  The empty sequence is used here as a kind of "null value".

Nodes: Tree Structure (1)

- Nodes form a tree, basically by `parent` and `children` properties.

  Also the `namespaces` and `attributes` properties, see below.

- Every node except document nodes can have a `parent`.

  Since the data model also permits document fragments, the `parent`-property can be empty (i.e. return the empty sequence). Otherwise, the value of the `parent`-property is a node of type `document` or `element`. The other node types can appear in the tree only as leaves.

- Document and element nodes can have `children`.

  This property is a list of `element`, `text`, `processing instruction`, and `comment` nodes.

Nodes: Tree Structure (2)

- Note that `attribute` and `namespace` nodes cannot appear in the `children` list, but they can have a `parent`.
- Nodes of these types are attached to their `parent` via specific properties:
  - ◇ The property `attributes` (of element nodes) is a sequence of `attribute` nodes.
  - ◇ The property `namespaces` (of element nodes) is a sequence of `namespace` nodes.

  Only `element` nodes have these properties.

Nodes: Tree Structure (3)

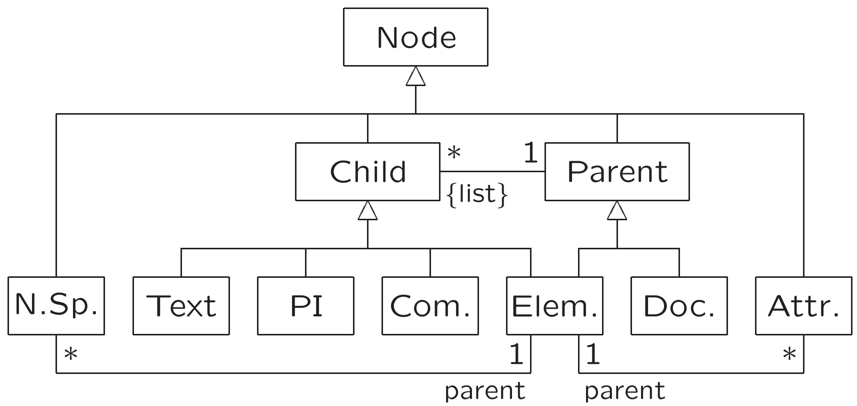
- The data model defines integrity constraints that ensure that the properties are consistent, e.g.
  - ◇ If a node X of type `element`, `text`, `comment`, or `processing instruction` has a node Y as `parent`, X must appear among the children of Y.
  - ◇ And vice versa: If X appears among the children of Y, then Y must be the parent of X.
  - ◇ Similar rules hold for `attribute`/`namespace` nodes.

Nodes: Tree Structure (4)

Node Kind	Can be parent	Can be Child
document	yes	no
element	yes	yes
attribute	no	no (see below)
text	no	yes
processing-instruction	no	yes
comment	no	yes
namespace	no	no (see below)

Note that "can be parent" (i.e. possibly appearing as value of the property "`parent`") is the same as having a property "`children`". In contrast, "can be child" (i.e. possibly appearing as value of the property "`children`") is not the same as having a property "`parent`". The exception are attribute and namespace nodes: They cannot be children, but have a parent. The converse link (from parent) is via the properties `attributes` and `namespaces`.

Nodes: Tree Structure (5)



Text Nodes

- The following constraints ensure that there are no superfluous text nodes:
  - ◇ The `children` property can never return a sequence that contains two consecutive text nodes.

    If two text nodes would appear directly one after the other, they must be merged.
  - ◇ The `children` property can never return a sequence that contains empty `text` nodes.

    Empty `text` nodes are permitted when they do not have a `parent`.

Whitespace (1)

- Whitespace between elements is a bit difficult:
  - ◇ It is often inserted for better readability, but it is not semantically important.

    E.g., when an element type is declared with pure element content, a validation is possible, even if there is additional whitespace between the elements.
  - ◇ But if an element has mixed content, even whitespace between elements might be important.
- Therefore, the XML standard specifies that such whitespace must be passed to the application.

  The application might then ignore it.

Whitespace (2)

- The XML Infoset standard specifies that character information items have an optional boolean property "element content whitespace".
- This is true for unimportant whitespace characters (appearing in elements with pure element content).
- Validating parsers must provide this property.
- If the XDM instance is constructed from an infoset that provides this property, text nodes are removed if they consist entirely of whitespace for which this property is true.

Whitespace (3)

- If a schema is used for validation (i.e. the XDM is constructed from a PSVI), text nodes that consist entirely of whitespace are removed if they are children of an element node whose "content-type" property is not `"mixed"`.
- In short:
  - ◇ If a validation is done (with respect to a DTD or a schema), there will be no text nodes for whitespace between element content.
  - ◇ If the XDM instance is built without validation, such text nodes are constructed.

Whitespace (4)

Some related Remarks:

- XSLT permits to define a set of elements for which pure whitespace child nodes should be removed:

```
<xsl:strip-space elements="..."/>
```

- With `xml:space="preserve"` one can specify in the XML data file that whitespace should be preserved.

  The other value is `"default"` which means that the application can do what it wants with the whitespace. This attribute is already introduced in the XML standard. For validating parsers, it must be declared.

Example (1)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Simple Example -->
<?xml-stylesheet type="text/xsl" href="ex.xsl">
<STUDENT SID="101">
    <FIRST>Ann</FIRST>
    <LAST>Smith</LAST>
</STUDENT>
```

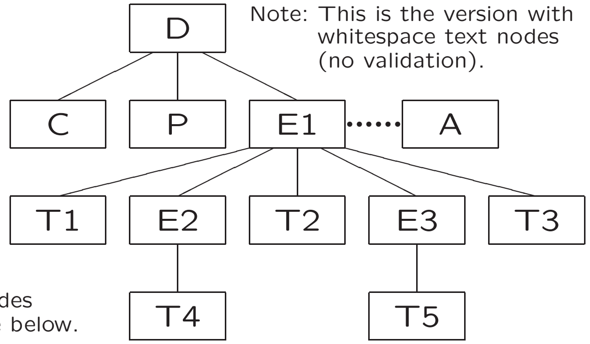
The stylesheet declaration is an example for a processing instruction. The stylesheet could e.g. translate these XML data to XHTML for display in a web browser.

Note that whitespace outside the document element (in this case, `STUDENT`) is already removed in the XML infoset from which the XDM instance can be constructed. Thus, although XDM permits text nodes as children of the document node, this is not used (if the instance is constructed this way).

Example (2)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Simple Example -->
<?xml-stylesheet type="text/xsl" href="ex.xsl">
<STUDENT SID="101">
    <FIRST>Ann</FIRST>
    <LAST>Smith</LAST>
</STUDENT>
```

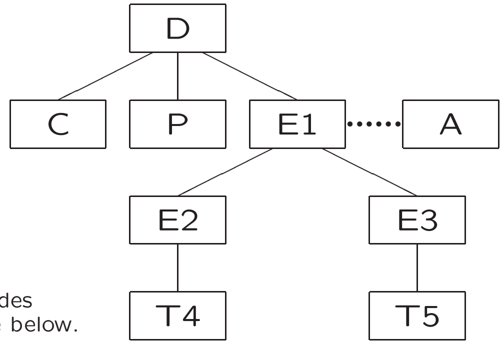
Note: This is the version with whitespace text nodes (no validation).



Note: Namespace nodes not shown, see below.

Example (3)

With validation, there are no whitespace text nodes:



Note: Namespace nodes not shown, see below.

Example (4)

- D: Document node (root).
  - ◇ `node-kind = "document"`
  - ◇ `children = (C, P, E1)`
- C: Comment node.
  - ◇ `node-kind = "comment"`
  - ◇ `parent = D`
- P: Processing instruction (Stylesheet information).
  - ◇ `node-kind = "processing-instruction"`
  - ◇ `parent = D`
  - ◇ `node-name = "xml-stylesheet"`
  - ◇ `string-value = 'type="text/xsl" href="ex.xsl"'`

Example (5)

- E1: Element node (STUDENT).
  - ◇ `node-kind = "element"`
  - ◇ `parent = D`
  - ◇ `children = (T1, E2, T2, E3, T3)`
  - ◇ `node-name = "STUDENT"`
  - ◇ `attributes = (A)`
- E2: Element node (FIRST).
  - ◇ `node-kind = "element"`
  - ◇ `parent = E1`
  - ◇ `children = (T4)`
  - ◇ `node-name = "FIRST"`

Example (6)

- E3: Element node (LAST).
  - ◇ `node-kind = "element"`
  - ◇ `parent = E1`
  - ◇ `children = (T5)`
  - ◇ `node-name = "LAST"`
- T1: Text node (whitespace after <STUDENT ...>).

  This node appears only if the XDM instance is constructed without validation (or if STUDENT has a mixed content model). The same applies to T2 and T3.

  - ◇ `node-kind = "text"`
  - ◇ `parent = E1`
  - ◇ `string-value = "\n    "` (note: \n is not XML syntax)

Example (7)

- T2: Text node (whitespace after </FIRST>).
  - ◇ `node-kind = "text"`
  - ◇ `parent = E1`
  - ◇ `string-value = "\n    "`
- T3: Text node (whitespace after </LAST>).
  - ◇ `node-kind = "text"`
  - ◇ `parent = E1`
  - ◇ `string-value = "\n"`
- T4: Text node (contents of <FIRST>).
  - ◇ `node-kind = "text"`
  - ◇ `parent = E2`
  - ◇ `string-value = "Ann"`

Example (8)

- T5: Text node (contents of `<LAST>`).
  - ◇ `node-kind = "text"`
  - ◇ `parent = E3`
  - ◇ `string-value = "Smith"`
- A: Attribute node (for `SID="101"`)
  - ◇ `node-kind = "attribute"`
  - ◇ `parent = E1`
  - ◇ `node-name = "SID"`
  - ◇ `string-value = "101"`
- In addition, there are three namespace nodes (one attached to each element node), see below.

Namespace Nodes (1)

- Although the example contains no explicit namespaces, the prefix `xml` is always bound to

  `http://www.w3.org/XML/1998/namespace`

- For each namespace declaration, a namespace node is attached to each element node that is in scope of that namespace declaration.

  I.e. not only to the element that explicitly contains the namespace declaration, but also to all descendants, as long as the same prefix is not bound to another URI (or to the empty URI which "undefines" the prefix).

Namespace Nodes (2)

- Namespace nodes cannot be shared between elements.

  They have a link to a specific element node in the `parent` property.

- Thus, the example contains already three namespace nodes.
- E.g. N1: Namespace node for `STUDENT`-Element:
  - ◇ `node-kind = "namespace"`
  - ◇ `parent = E1`
  - ◇ `node-name = "xml"`
  - ◇ `string-value = "http://www.w3.org/XML/1998/namespace"`

Namespace Nodes (3)

- Namespace nodes are accessible in XPath 1.0 by the namespace axis.
  - ◇ In XPath 2.0, use of the namespace axis is deprecated. In XQuery 1.0, it does not exist.
  - ◇ Instead, one should use XPath functions.
  - ◇ These functions do not permit access to the node identity or parent node of a namespace node.
  - ◇ Then, namespace nodes can be shared between element nodes.

Namespace Nodes (4)

- Because of the problem with namespace nodes, XDM has two alternative accessor functions (both correspond to the property "`namespaces`"):
  - ◇ `namespace-nodes`: This returns a sequence of namespace nodes.

    If namespace nodes are not needed, this accessor does not have to be implemented.
  - ◇ `namespace-bindings`: This returns the namespace declarations valid at an element node as a set of prefix/URI pairs.

    The standard says that the representation is implementation-dependent, but declares the return type as sequence of `xs:string`.

Document Order (1)

- The document order is a total order on nodes.
- Within a tree, the root node is the first node.

  This actually follows from the other rules.
- Every node occurs before all its children and descendants.
- The relative order of siblings is the order in which they occur in the `children` property of their parent.
- Children and descendants of a node occur before following siblings.

Document Order (2)

- Namespace nodes immediately follow the element node with which they are associated.

  The relative order of namespace nodes is implementation defined, but stable (i.e. if two namespace nodes of the same element node are compared several times, the result is always the same).
- After the namespace nodes, (or the element node, if there are no namespace nodes), the attribute nodes immediately follow.

  The relative order of attribute nodes is implementation defined, but stable.

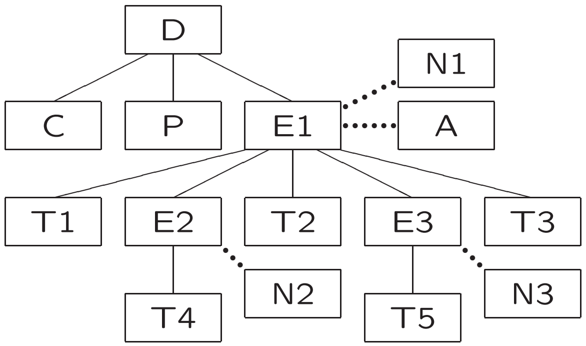
Document Order (3)

- Alternative definitions:
  - ◇ The document order is simply the sequence of a pre-order traversal of the tree, with the namespace and attribute nodes listed immediately after their element node.
  - ◇ The document order is simply the order of the begin of the start of a node value in the XML document (assuming that namespaces are defined before other attributes).

Document Order (4)

- The relative order of nodes of different trees is implementation-defined (but stable) with the following restriction:
  - ◇ If one node of tree T_1 appears between one node of tree T_2, all nodes of tree T_1 must appear before all nodes of tree T_2.
- I.e. the document order on the nodes of several trees can be derived from some order on the trees and the order of the nodes within each tree.

Document Order (5)



- Document order: D, C, P, E1, N1, A, T1, E2, N2, T4, T2, E3, N3, T5, T3 (unique in this example).

Exercise

Please draw the XDM tree:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!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>My first XHTML document</title>
    </head>
    <body>
        <h1>Greeting</h1>
        <p>Hi, <a href="http://www.w3.org">W3C</a>!</p>
    </body>
</html>
```

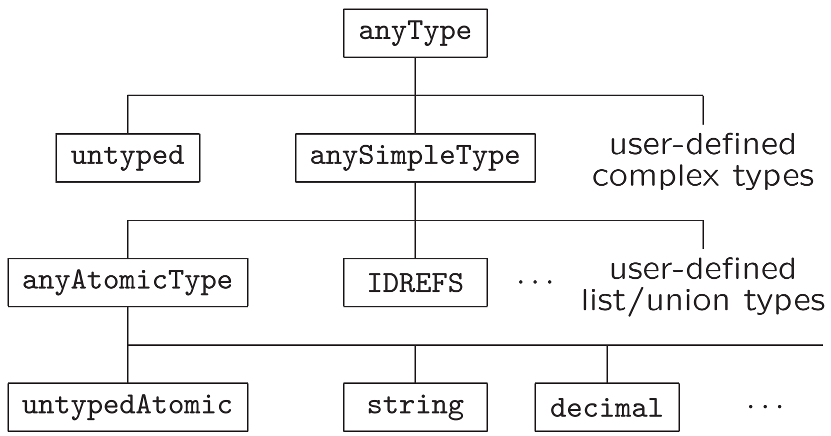

String/Typed Value (1)

- Three important accessor functions are:
  - ◇ `string-value`, with return type `xs:string`.
  - ◇ `typed-value`, with static (declared) return type `xs:anyAtomicType*` (a sequence of atomic values).

    The dynamic type of the actually returned values may be more specific. The sequence that can possibly be returned is used for list types.
  - ◇ `type-name`, with return type `QName?` (i.e. a qualified name or the empty sequence).

    This gives the real (dynamic) type of the value that `typed-value` returns. Remember that atomic values have a type attached.

String/Typed Value (2)



String/Typed Value (3)

- Document, element, and attribute nodes have properties “string-value”, “typed-value”, “type-name”.
- However, the corresponding accessor functions are also important for the other node kinds:
  - ◇ For text, comment, and processing-instruction nodes, they return the value of the “content” property,
  - ◇ for namespace nodes, they return the value of the “uri” property.

    For these four kinds of nodes, the `typed-value` is the same as the string-value, the dynamic type of `typed-value` is `xs:string`.

String/Typed Value (4)

- For document nodes,
  - ◇ the string-value is simply the content of all text nodes that are descendants of the document node, concatenated in document order.

    I don’t know why this is a property and not simply computed by the accessor function.
  - ◇ The typed-value is the same, but with the type `xs:untypedAtomic`.
  - ◇ In the above example, `string-value` of the document node is `"AnnSmith"` (if validation is done) or `"\n    Ann\n    Smith\n"` (without validation).

String/Typed Value (5)

- For an attribute node,
  - ◇ `string-value` is the normalized attribute value.

    The normalization is defined in the XML standard (Section 3.3.3). Basically, all whitespace characters are translated into space characters (CR-LF line ends are translated to a single space). If the data type of the attribute is known and is different from `CDATA`, also leading and trailing spaces are removed, and sequences of space characters are translated into a single space. If a schema is used for validation, the schema normalized value is used (see facet `whiteSpace`). In this case, any lexical representation of the typed value can be returned (see below).
  - ◇ If no schema is avaliable, the `typed-value` is the `string-value` as an `xs:untypedAtomic`.

String/Typed Value (6)

- Attribute nodes, continued:
  - ◇ If the XDM instance is constructed from a PSVI (i.e. validation with respect to an XML schema was done), the `typed-value` is the attribute value converted to the type declared for that attribute.

    If the document is invalid or only partially validated, the type is `xs:anySimpleType`. If the validation was not attempted, or the result is not known, the type is `xs:untypedAtomic`.

    In case of a union type, the value is converted to the first member type of the union for which it is valid. In case of a list type, the value is a sequence of the atomic list members.
  - ◇ The property `type-name` returns this type.

    For union types, it is the this type, not the actual member type.

String/Typed Value (7)

- String value of element nodes:
  - ◇ If the element is declared with a simple type as content, the `string-value` is the schema normalized value of the concatenation of all text node children.

    If a schema is used, elements with a simple type as content are treated equivalently to attributes.

    Actually, any lexical representation of the typed value can be returned (see below).
  - ◇ Otherwise, it is the concatenation of all descandant text nodes (in document order).

String/Typed Value (8)

- Typed value of element nodes:
  - ◇ If the element is declared with a simple type as content, the typed-value is the string-value converted to the declared/actual type.

    Because of `xsi:type`, there can be a difference between declared and actual type (also for union types?). The standard needs one page (Section 3.3.1.1) to explain what type is chosen. This case (the element corresponds to an attribute) is the only interesting case (the typed value is defined and different from string value).
  - ◇ If the element is declared as empty element, its `typed-value` is the empty sequence.

    Note that if it is declared e.g. with mixed content, but happens to have empty content, the typed-value is the empty string.

String/Typed Value (9)

- Typed value of element nodes, continued:
  - ◇ If the element has mixed content or its type is unknown (includes the construction from infoset, i.e. no validation or DTD-validation) the typed-value is the string-value as `xs:untypedAtomic`.
  - ◇ If the element is declared with pure element content, its `typed-value` is undefined and trying to access it raises an error.
- Also in these cases, the type-name is defined.

  In the usual case, it is the type declared in the schema for this element type (or `xs:untyped` if there is no schema).

String/Typed Value (10)

- An implementation may store the string-value, the typed-value, or both.
- If it stores only the typed-value, it may return any lexical representation of this value as string value.

  For example, suppose that `SID` is declared as `integer`. Consider the input: `<STUDENT SID="  00101">...</STUDENT>`. The typed-value of the attribute `SID` is the integer 101. An implementation may return `"101"` or `"00101"` as string-value (or other equivalent representations).
- If it stores only the string-value, it must convert the string to the correct member union type.

  Although type-name is probably only the union type. Also determining the namespace URI for QName and NOTATION might not be trivial.

is-id/is-idrefs

- If there is only a DTD (no schema), `type-name` is `xs:untypedAtomic` even for `ID`/`IDREF`(`S`)-attributes.

  Even if there is no DTD, one can call an attribute `xml:id` to make clear that it is a unique ID of the node.
- But these attributes are of special importance.
- Thus, boolean properties `is-id` and `is-idrefs` were introduced to mark such attributes.

  There is no property `is-idref` because a value in XDM is always a sequence (possibly of length one). The property `is-idrefs` is true when the attribute type is `IDREF` or `IDREFS`. The property `is-id` is true when the atrribute is of type `ID` or is called `xml:id`. When there is a schema, these properties can be true also for element nodes.

Nilled

- In XML Schema, `xsi:nil="true"` was introduced to mark elements that have a NIL/Null value (different from the empty content).

  The element type must be declared as `nillable`.
- The XDM property "nilled" is true for element nodes when the node was validated according to a schema and `xsi:nil="true"` was used.
- The typed-value of nilled element nodes is the empty sequence.

  The type-name is not changed (it is the type declared for the element).

Base URI (1)

- Document, element, and processing instruction nodes have a base URI property that can be used for resolving relative URIs in them.

  The base URI might be different for different parts of the document tree in case external entities were expanded (or because of `xml:base`, see below).
- The base URI is usually the URI of the input document (or external entity). However, the value of an `xml:base` attribute takes precedence.
- The URI might contain characters that must be escaped if a lexical/external representation is needed.

Base URI (2)

- If a processing instruction has a base URI different from its parent, it is difficult/impossible to keep this in the external representation.

  For all other nodes, the `xml:base` attribute can be used. In this case, one would have to write to that URI, which might be impossible or at least unwanted. The problem is that one cannot use `xml:base` in processing instructions.
  This shows that the XML standards do not fit completely together. (newer standards must live with design decisions done in older standards, already in the SGML standard). Things would probably become simpler and more consistent if a complete redesign were done).

- The document node has also a document-uri.

  This is an absolute URI that should be used to reload the document if necessary.

Nodes: Summary (1)

- Below, the properties of each node kind are listed.
- Unless otherwise noted, there is an accessor function for each property with the same name.
- In addition, there is an accessor function `node-kind` that returns the “subclass name” (see above).
- For attribute, comment, and text nodes, the accessor function `base-uri` returns the `base-uri` of its parent (if it exists, otherwise the empty sequence).
- All other accessor functions return the empty sequence.

Nodes: Summary (2)

Properties of Document Nodes:

- base-uri (possibly empty)
- children (possibly empty)
- unparsed-entities (possibly empty)

  This property is queried with the functions `unparsed-entity-public-id` and `unparsed-entity-system-id`. In contrast to all other accessor functions, which have a single parameter for the node, these functions have two parameters: One for the node and one for the entity name.

- document-uri (possibly empty)
- string-value
- typed-value

Nodes: Summary (3)

Properties of Element Nodes:

- base-uri (possibly empty)
- node-name
- parent (possibly empty)
- type-name
- children (possibly empty)
- attributes (possibly empty)
- ...(continued on next slide)

Nodes: Summary (4)

Properties of Element Nodes, continued:

- namespaces

  As explained above, `namespaces` is the name of the property, and `namespace-nodes` and `namespace-bindings` are two alternative accessor functions corresponding to this property.

- nilled
- string-value
- typed-value
- is-id
- is-idrefs

Nodes: Summary (5)

Properties of Attribute Nodes:

- node-name
- parent (possibly empty)
- type-name
- string-value
- typed-value
- is-id
- is-idrefs

Nodes: Summary (6)

Properties of Namespace Nodes:

- prefix (possibly empty)

  If the prefix is not empty, it is returned by the `node-name` accessor function (as `QName` with empty prefix and empty namespace URI). If the prefix is empty (for a default namespace declaration), `node-name` returns the empty sequence.

- uri

  This property is returned by the `string-value` and `typed-value` accessor functions. The type of the `typed-value` is `xs:string`. The accessor function `type-name` returns the empty sequence.

- parent (possibly empty)

Nodes: Summary (7)

Properties of Processing Instruction Nodes:

- target

  This property is returned by the `node-name` accessor function (as `QName` with empty prefix and empty namespace URI).

- content

  This property is returned by the `string-value` and `typed-value` accessor functions. The type of the `typed-value` is `xs:string`. The accessor function `type-name` returns the empty sequence. Note that the content must not contain "`?>`".

- base-uri (possibly empty)
- parent (possibly empty)

Nodes: Summary (8)

Properties of Comment Nodes:

- content

  This property is returned by the `string-value` and `typed-value` accessor functions. The type of the `typed-value` is `xs:string`. The accessor function `type-name` returns the empty sequence. Note that the content must not contain "--" and must not end in "-".

- parent (possibly empty)

Nodes: Summary (9)

Properties of Text Nodes:

- content

  This property is returned by the `string-value` and `typed-value` accessor functions. The type of the `typed-value` is `xs:string`. The accessor function `type-name` returns the empty sequence.

- parent (possibly empty)