

Introduction to XQuery

Srinivas Pandrangi
Chetan Patel

Ipedo Webinar Series

The logo for IPEDO, featuring the word "IPEDO" in a bold, black, sans-serif font. Above the letters "P" and "E" are three horizontal red lines of varying lengths, creating a stylized graphic element.

IPEDO

Agenda

- Background
- Basics
- XQuery Expressions
- Q & A

A brief history

1999	W3C XML Query group chartered (Oct)
2000	<ul style="list-style-type: none">• WG published Requirements doc (Jan)• <i>XQuery Data Model</i> (May)• <i>XML Query Algebra</i> (Dec)
2001	<ul style="list-style-type: none">• WG published first draft of XQuery1.0 (Feb)• <i>Functions & Operators</i> (Aug)
2003	<ul style="list-style-type: none">• <i>Data Model</i> and <i>F&O</i> in last call• Last round of publications in May

Related Standards

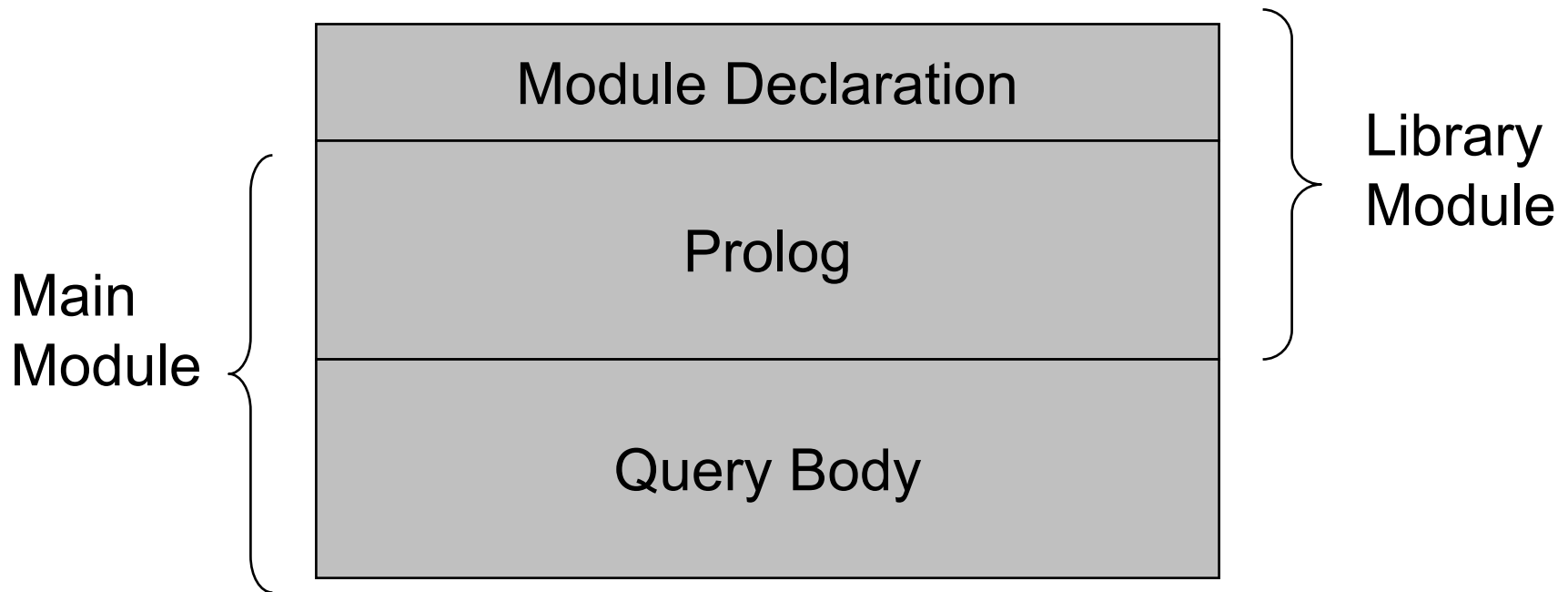
XPath 2.0	Developed jointly by XQuery and XSLT WGs
XML Schema	Type system aligned with XML Schema
Infoset	Data Model derived from Infoset (PSVI)

Part II - Basics

- Structure of Queries
- The XQuery Data Model

Structure of a Query

An XQuery query is composed of one or more **modules**



Structure of a Query

A sample **Main Module**:

```
declare namespace ns = "http://www.example.com"  
declare function factorial($n as xs:integer) as xs:integer {  
    if($n = 0)  
    then 1  
    else ($n * factorial($n-1))  
}
```

— **Prolog**

```
factorial(10)
```

Body (expression)

Structure of a Query – contd.

A sample **Library Module**

Module Declaration

```
module namespace math = "http://www.example.com/math"
```

```
declare function math:factorial($n as xs:integer) as xs:integer {  
    if($n = 0)  
    then 1  
    else ($n * factorial($n-1))  
}
```

```
declare function math:square($n as xs:integer) as xs:integer {  
    $n * $n  
}
```

Prolog

Prolog

The Prolog sets the environment for the query:

- Namespace declarations

```
eg: declare namespace ns = "http://www.example.com"
```

- Default element and function namespace declarations

```
eg: declare default element namespace = "http://www.example.com"
```

- Variable declarations

```
eg: declare variable $ctr {1}
```

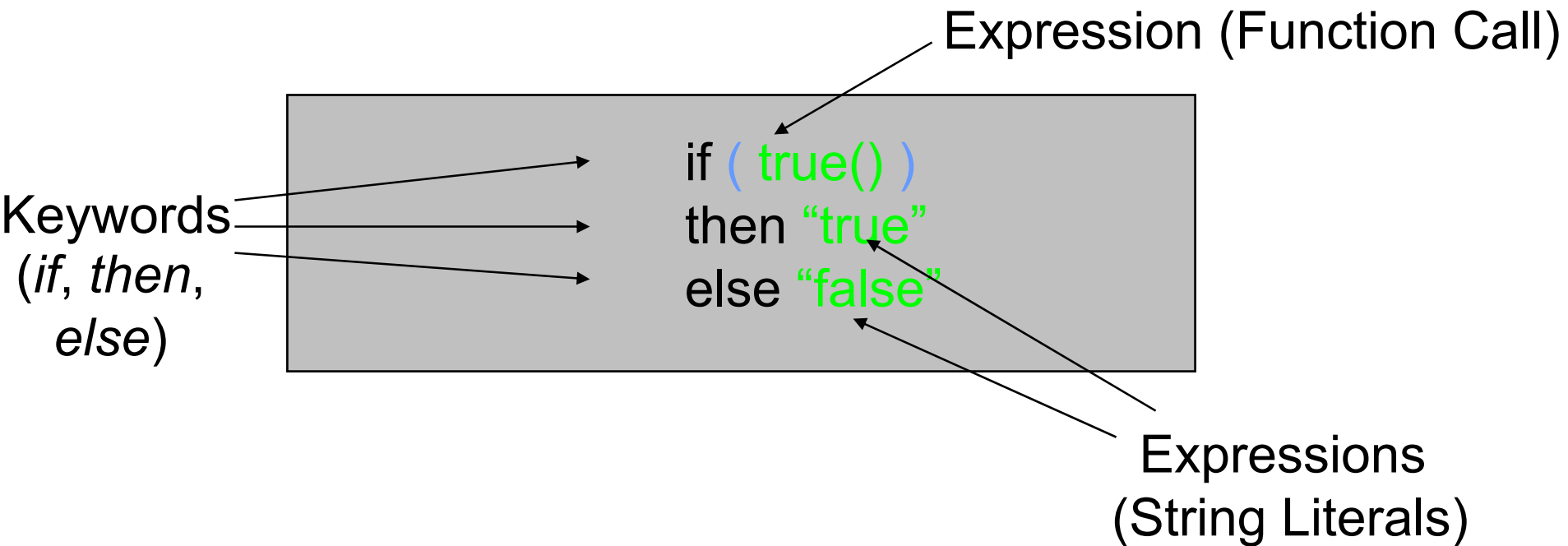
- Module imports

```
eg: import module namespace math="http://www.example.com/math"
```

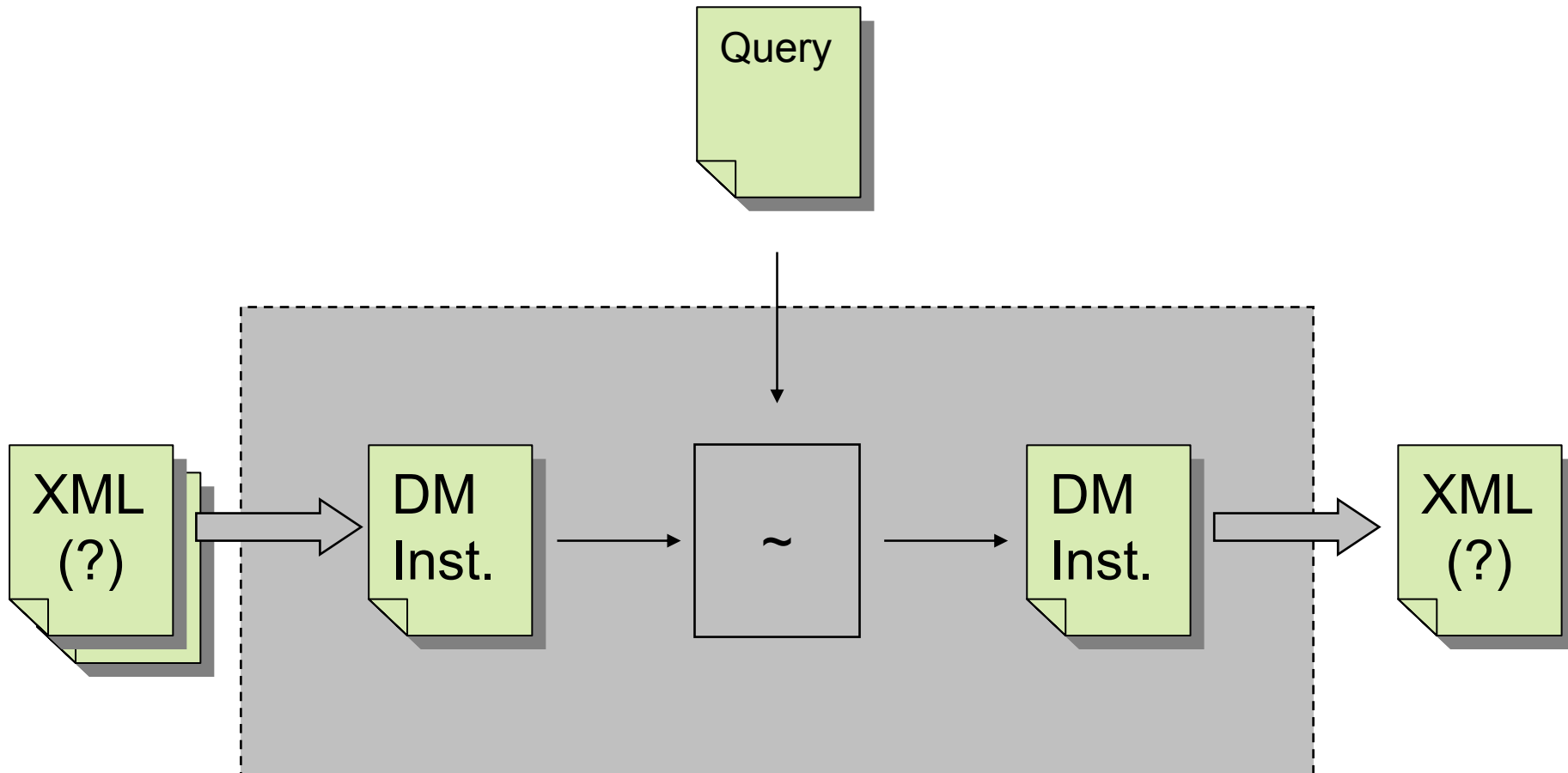
- User defined function definitions

Expressions

- XQuery is a functional language
 - Expressions are the building blocks of queries.
 - Expressions can be composed from keywords, symbols and other expressions



XQuery Processing



Data Model

The data model describes all the inputs to, and the outputs from an XQuery processor. It also describes intermediate values

Value categories are:

- Sequence
 - Items
 - Nodes
 - Document
 - Element
 - Attribute
 - Text
 - Comment
 - Processing Instruction
 - Namespace
 - Atomic Values

XML Document Instance

<!-- List of books recommended this month by the Young Readers Book Club -->

<books>

<book isbn="0060929871">

<title>A Brave New World</title>

<author>Aldous Huxley</author>

<genre>science fiction</genre>

<price>14.25</price>

</book>

<book isbn="0590353403">

<title>Harry Potter and the Sorcerer's stone</title>

<author>JK Rowling</authors>

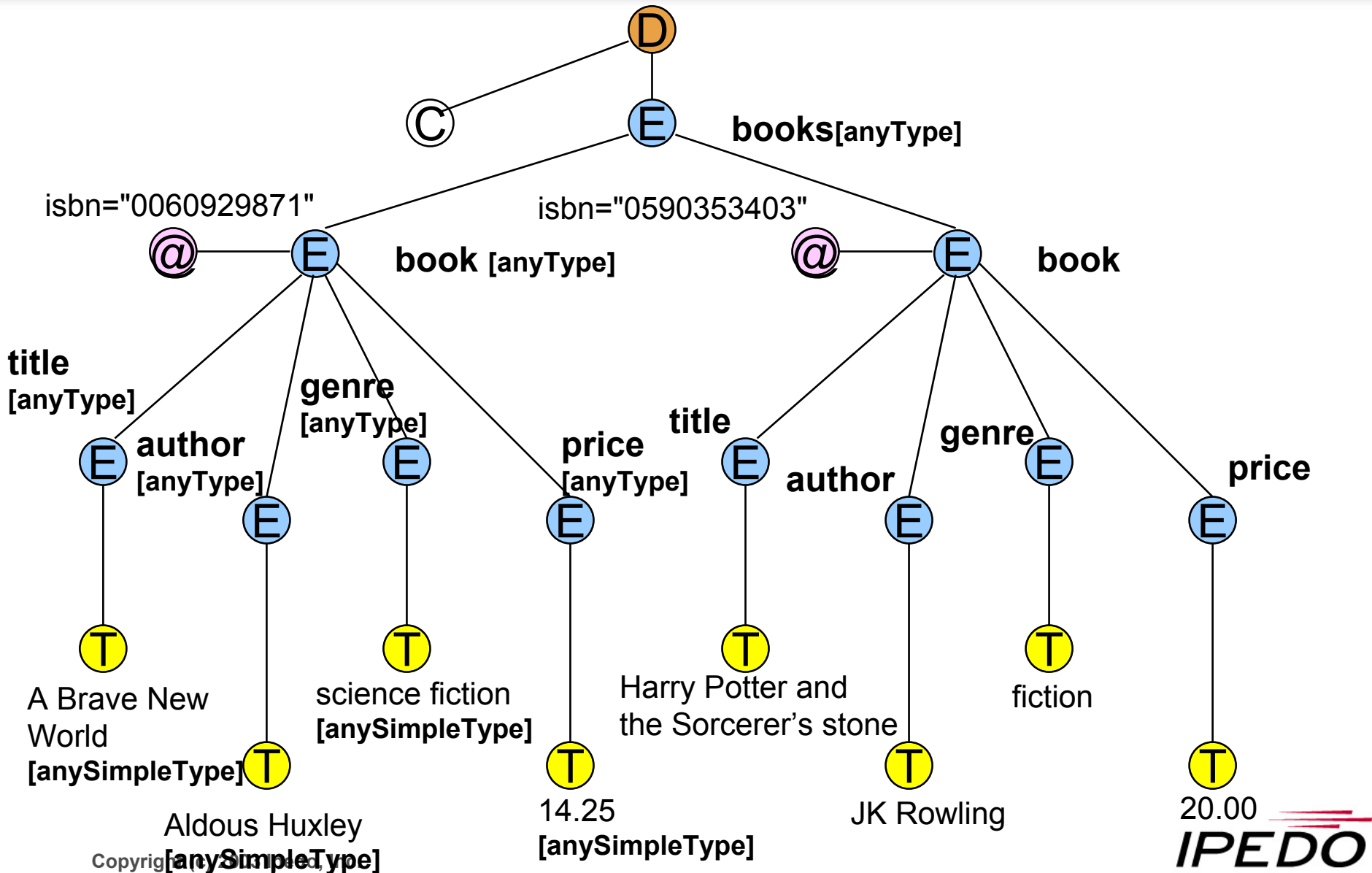
<genre>fiction</genre>

<price>20.00</price>

</book>

</books>

XQuery Data Model Instance



Part III – XQuery Expressions

- Path Expressions
- Constructors
- FLWOR Expressions
- Arithmetic and Logical Expressions
- Conditional Expressions
- Quantified Expressions
- Functions (XQuery Function Library & UDFs)
- Expressions on Data Types

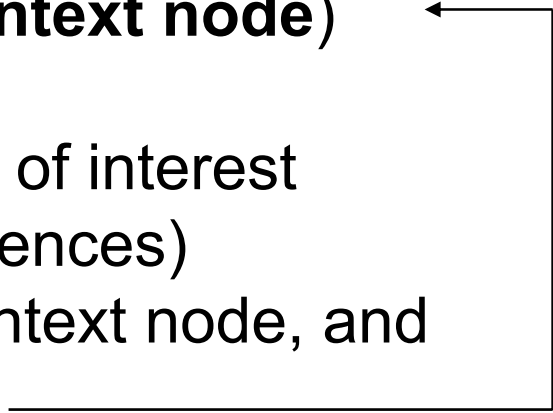
Primary Expressions

- Primary Expressions are the most basic types of expressions
 - Literals
 - String literals – eg: “Hello World”
 - Numeric literals – eg: 127.3, 1.23E3
 - Variables
 - eg: \$name
 - Function Calls

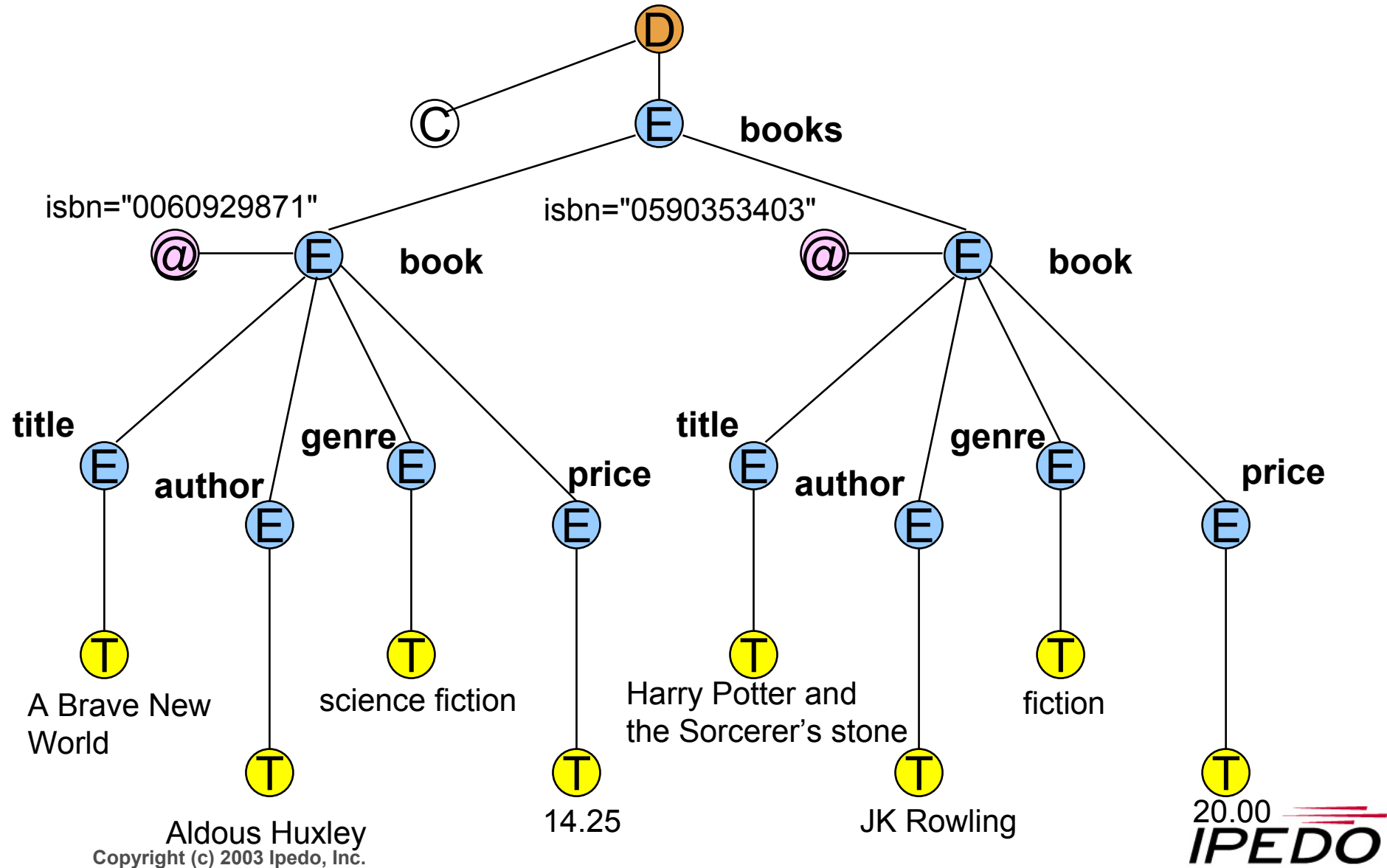
Path Expressions

Path expressions allow you to trace a path through an XML document tree to the nodes you want to extract

Path Expression Processing:

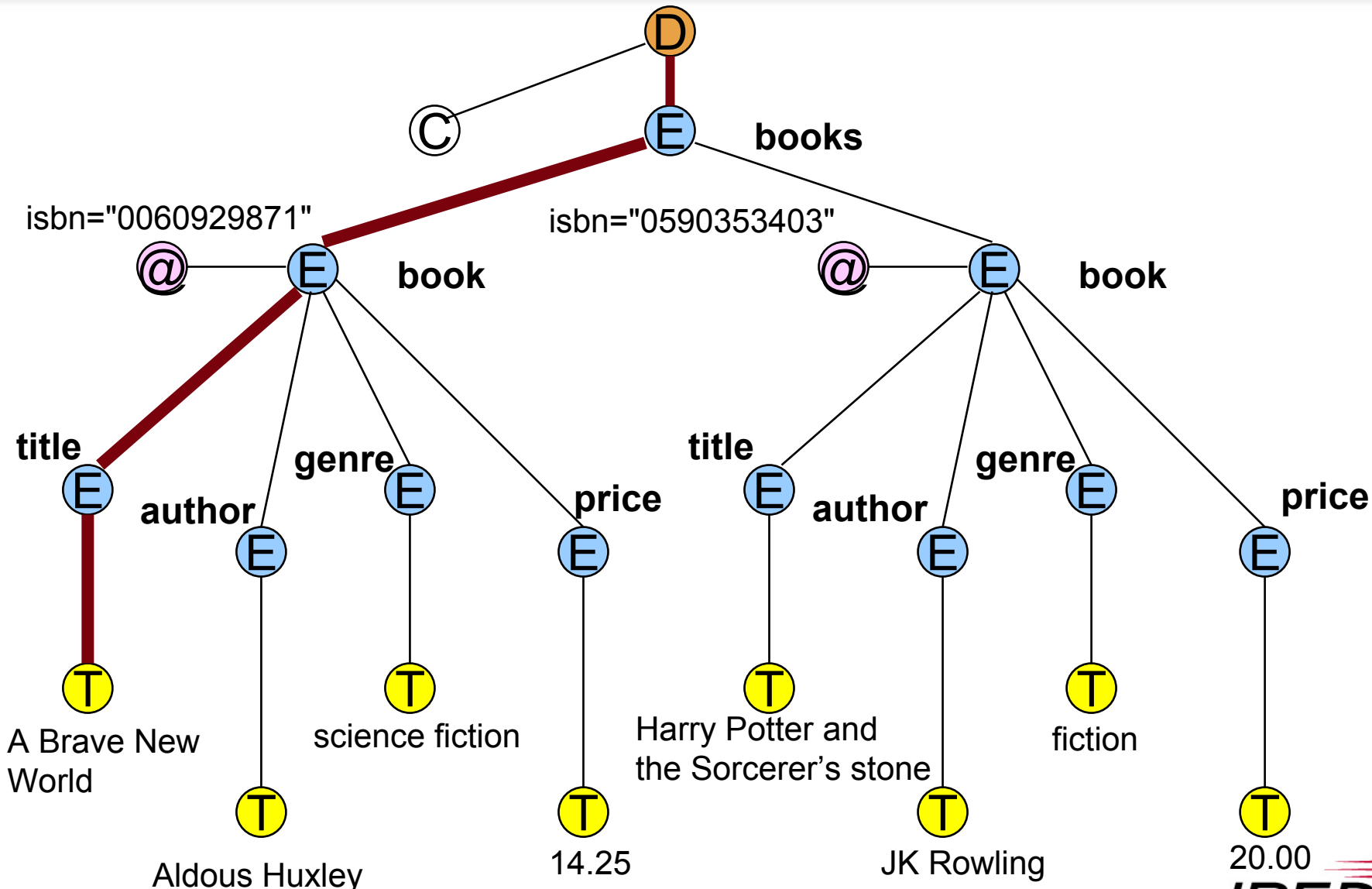
- Pick a **starting location** in the tree (**context node**)
 - Pick a **direction** to take (**axis**)
 - Pick a **condition(s)** for selecting nodes of interest (kindtest, nametest, predicates, dereferences)
 - For each selected node, make it the context node, and repeat
- 

Data Model Instance

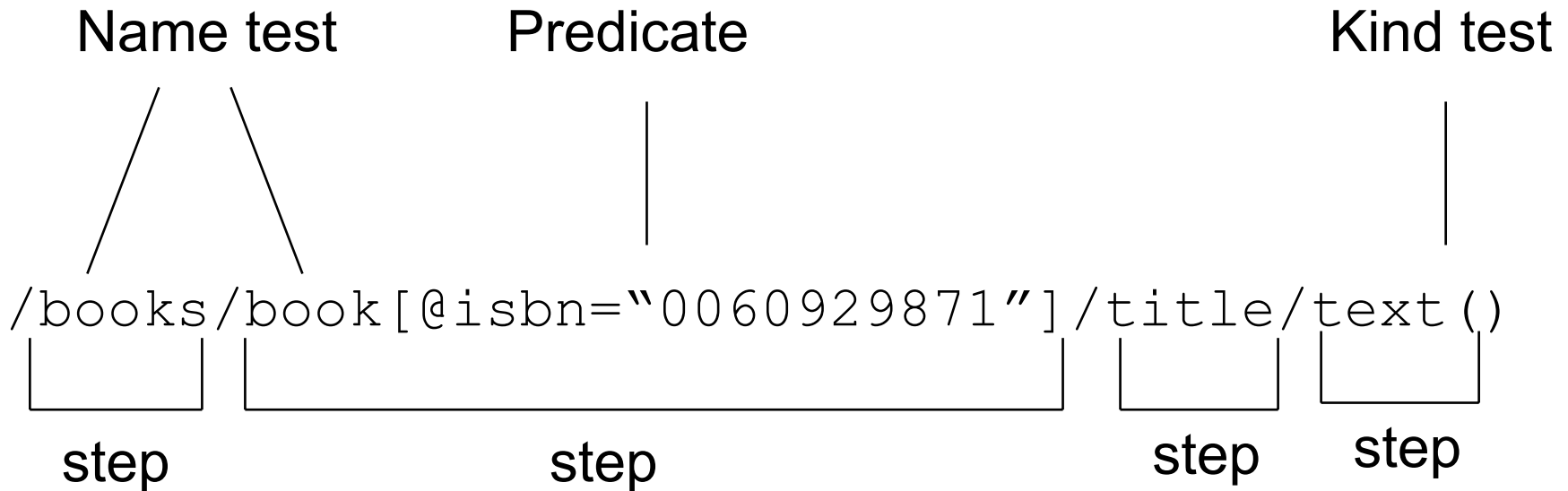


Get the title of the book with the ISBN# 0060929871

```
/books/book[@isbn="0060929871"]/title/text()
```



Expression Evaluation



Constructors

Used to construct elements/attributes/comments/Pis/CData Sections

- Direct Element and attribute constructors:
 - Uses an XML syntax

```
<greeting>Hello World</greeting>
```

- Dynamic element content through expressions

```
<greeting>Hello {$name}</greeting>
```

- Attributes are constructed similarly

```
<greeting type="hello">Hello {$name}</greeting>
```

```
<greeting type="{greetingType}">Hello {$name}</greeting>
```

Computed Constructors

- Computed element/attribute/document/comment/text/ns constructors
 - A keyword based syntax for constructing elements/attributes/...
 - Used when element/attribute names are computed (not constant)

Syntax:

```
element name|expression { expression }
```

```
_____
```

Keyword

Element name

Element content

eg:

```
element greeting { $greeting }
```

```
element $tagname { $greeting }
```

Attributes use a similar syntax

```
element $tagname { attribute $attrname { $attrValue } }
```

FLWOR Expressions

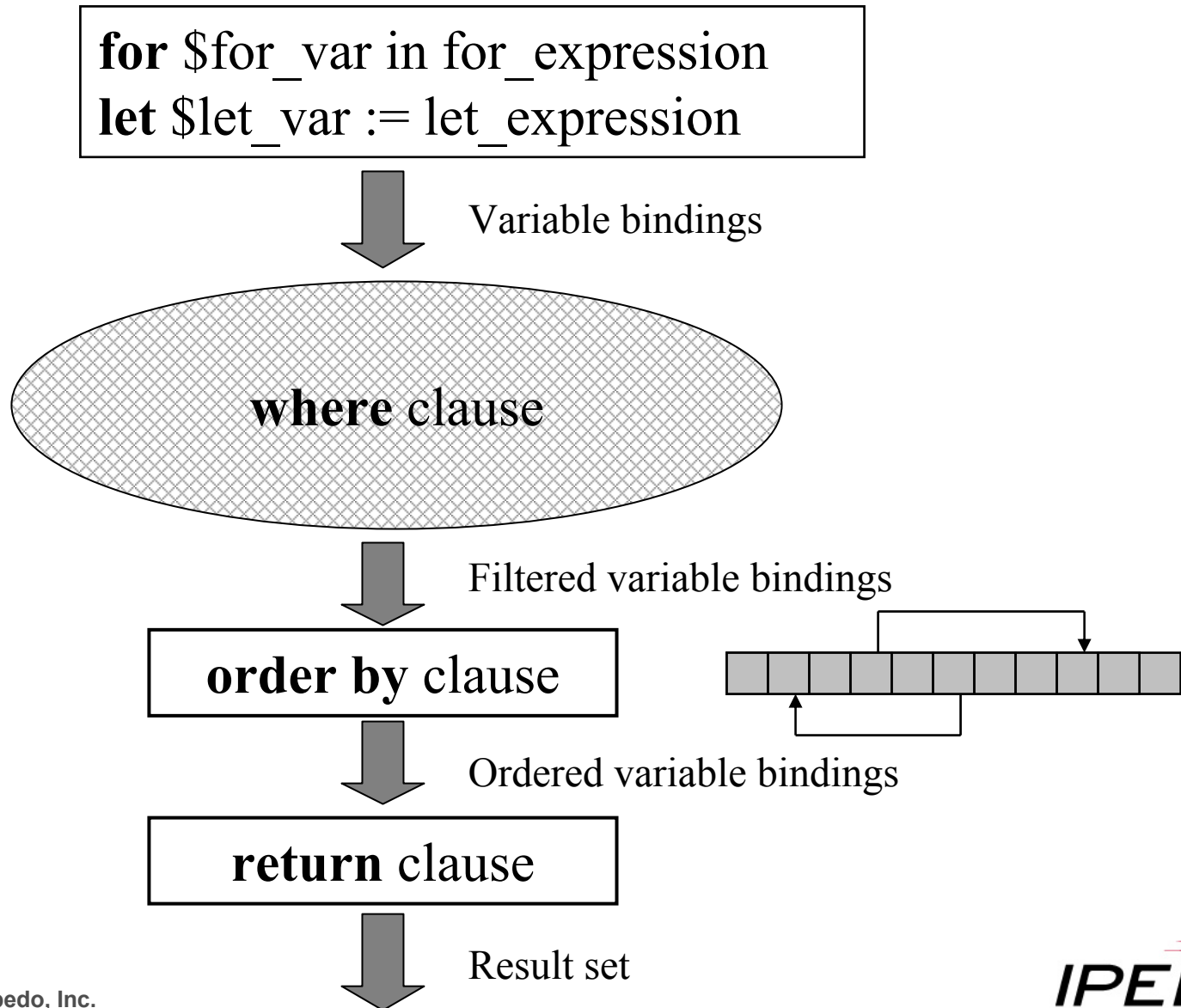
The name FLWOR (pronounced “flower”) stands for the five clauses **f**or, **l**et, **w**here, **o**rders by and **r**eturn, that make up these expressions.

Syntax:

```
for <for_var> in for_expression          [...]  
let <let_var> := let_expression         [...]  
where <filter_expression>  
order by <order_spec>  
return <expression>
```

- **for**: creates n bindings for the variable *for_var*
- **let**: creates 1 binding for the variable *let_var*
- **where**: filters the bindings using the *filter_expression*
- **order by**: orders the bindings based on *order_specs*
- **return**: clause is executed once for every surviving binding

FLWOR Expressions – Contd.



FLWOR Expressions – Contd.

Illustration:

Return all the books published before the year 1950

```
<old_books>
{
    for $book in document("catalog/books.xml")/books/book
    where $book/year_published < 1950
    order by $book/year_published ascending
    return $book
}
</old_books>
```

FLWOR Expressions - Joins

FLWOR expressions allow for joining content of two or more documents

e.g.:

Return inventory information for all books

```
declare namespace i = "http://www.books.com/inventory"
for $book in document("catalog/books.xml")/books/book,
  $book_inventory in document("catalog/inventory.xml")/i:inventory/i:item
where $book/@isbn = $book_inventory/@id
return
<book_info>
{
  $book/title, $book/publisher, $book_inventory/i:qty_in_stock
}
</book_info>
```

FLWOR Expressions - Grouping

Grouping can be achieved by nesting for clauses, or by using let

eg:

Return all the books with all reviews of books grouped under each book

```
for $book in document("catalog/books.xml")/books/book
return
<book_info>
{
  $book/title,
  for $review in document("catalog/reviews.xml")/book_reviews/review
  where $review/@bookid = $book/@isbn
  return $review
}
</book_info>
```

Arithmetic and Logical Expressions

Arithmetic expressions

- Binary operators: **+**, **-**, *****, **div**, **idiv** and **mod**
- Unary operators: **+**, **-**
- Operators overloaded for operands of different types

Comparison expressions

- **Value Comparisons**
 - Operators: **eq**, **ne**, **le**, **lt**, **ge**, **gt**
 - Compare single values
- **General Comparisons**
 - Operators: **=**, **!=**, **<**, **<=**, **>**, **>=**
 - Apply to sequences
 - Use existential semantics i.e, (**\$book/genre = "fiction"**) is true if there exists a genre element whose value is equal to "fiction"

Comparison Expressions – Contd.

- **Node Comparisons**

- Operators: *is*, *isnot*
- Apply to nodes; compare identity
- Both operands must either be a single node or an empty sequence
- *expr1 is expr2* is true if the evaluation of *expr1* results in a node which has the same identity as the result of evaluating *expr2*

- **Order comparisons**

- Operators: <<, >>
- Apply to nodes; compare document order
- *node1 << node2* is true if node1 is before node2 in document order

Logical Expressions

Logical Operators “**and**” and “**or**”:

- The concept of **Effective Boolean Value(EBV)** is key to evaluating logical expressions:

EBV of an empty sequence is *false*

EBV of a non-empty sequence containing only nodes is *true*

EBV is the value of the expression if the expression evaluates to a value of type `xs:boolean`

EBV is an error in every other case.

eg: The expression “`() and true()`” evaluates to *false* (since `()` is false)

A `not()` function is implemented which returns a negation of the *EBV* of the parameter

Conditional Expressions

Syntax:

```
if (expr1) then expr2 else expr3
```

if *EBV* of *expr1* is true, the conditional expression evaluates to the value of *expr2*, else it evaluates to the value of *expr3*

eg: Alternative way of writing the FLWOR example

```
<cheap_books>
```

```
for $book in document("catalog/books.xml")/books/book
```

```
return
```

```
    If ($book/price < 10)
```

```
    then $book
```

```
    else ()
```

```
</cheap_books>
```

Quantified Expressions

Syntax:

[some | every] $\$var$ in $expr$ satisfies $test_expr$

Quantified expressions evaluate to a boolean value.

Evaluation:

- $\$var$ is bound to each of the items in the sequence resulting from $expr$
- For each binding, the $test_expr$ is evaluated
- In case of
 - Existential quantification (“some”), if atleast one evaluation of $test_expr$ evaluates “true”, the entire expression evaluates “true”
 - Universal quantification (“every”), all evaluations of $test_expr$ must result in an EBV of “true” for the expression to return “true”

Quantified Expressions – Contd.

e.g.:

(: Find out if a particular book has at least one good review :)

if(some \$review in

document("catalog/reviews.xml")/book_reviews/reviews/review

[@bookid="0060929871"]

satisfies \$review/@rating > 3)

then "At least one good review"

else "All reviews are sub-par"

Input
Expr

Test
Expr

The XQuery Function Library

The XQuery1.0 & XPath2.0 Functions & Operators draft defines over 200 functions and operators.

Categories:

- Constructors – for constructing values of simple types (string, integer, decimal, double, date etc.)
- Func and operators on strings – compare, concat, substring etc.
- Func and operators on numbers – floor, ceiling, round etc.
- Func on Date/Time types – getXYZFromDate/Time, date arithmetic
- Func on Nodes – copy, name, local-name, deep-equals etc.
- Func on Sequences – concatenate, empty, insert, remove, distinct-values
 - Operations – union, intersect, except
 - Aggregate Functions – count, max, min, avg
- Input functions – document, collection, input
- Context functions – position, context-item etc.
- Accessors – base-uri, data, node-kind etc.

User Defined Functions

Syntax:

```
declare function funcName ($param1 as paramType1, ....) as returnType {  
    functionBody (an expression)  
}
```

paramType and *returnType* are instances of a *sequenceType*

eg:

```
declare function calculateDiscount($price as xs:double) as xs:double {  
    (: some expression that calculates the discount :)  
    $price * 0.1  
}
```

The Specifications

- XQuery1.0/XPath2.0 Data Model
- XQuery1.0 Formal Semantics
- XQuery1.0 – An XML Query Language
- XML Path Language – XPath2.0
- XQuery1.0/XPath2.0 Functions and Operators
- XQuery 1.0/XSLT2.0 Serialization
- XML Query use cases

Public drafts are available at <http://www.w3c.org/XML/Query>

Comments can be sent to public-qt-comments@w3.org

Comments are archived at

<http://lists.w3.org/Archives/Public/public-qt-comments>

Book Drawing

Three winners are picked at random.



Next Steps

- Update expressions
- Views
- Prepared Statements
- Stored Procedures
- ???

Check out the article “Five Practical applications of XQuery” [<http://www.devx.com/xml/Article/15618/>], for sample apps

spandran@ipedo.com

Try Ipedo's XQuery Implementation. Download the eval Edition of Ipedo XML Information Hub v3.3 at <http://www.ipedo.com/developer/>