Structured Web Documents in XML

Bojan Furlan

A Semantic Web Primer,
G. Antoniou, F. van Harmelen

An HTML Example

```
<h2>Nonmonotonic Reasoning: Context-
   Dependent Reasoning</h2>
<i>by <b>V. Marek</b> and
   <b>M. Truszczynski</b></i><br>
Springer 1993<br>
ISBN 0387976892
```

The Same Example in XML

```
<book>
    <title>Nonmonotonic Reasoning: Context-
        Dependent Reasoning</title>
    <author>V. Marek</author>
    <author>M. Truszczynski</author>
    <publisher>Springer</publisher>
    <year>1993</year>
    <ISBN>0387976892</ISBN>
</book>
```

HTML versus XML: Similarities

- Both use tags (e.g. <h2> and </year>)
- Tags may be nested (tags within tags)
- Human users can read and interpret both HTML and XML representations quite easily

… But how about machines?

Problems with Automated Interpretation of HTML Documents

An intelligent agent trying to retrieve the names of the authors of the book

- Authors' names could appear immediately after the title
- or immediately after the word by
- Are there two authors?
- Or just one, called "V. Marek and M. Truszczynski"?

HTML vs XML: Structural Information

- HTML documents do not contain structural information: pieces of the document and their relationships.
- XML more easily accessible to machines because
  - Every piece of information is described.
  - Relations are also defined through the nesting structure.
  - E.g., the **<author>** tags appear within the **<book>** tags, so they describe properties of the particular book.

HTML vs XML: Structural Information (2)

- A machine processing the XML document would be able to deduce that
  - the **author** element refers to the enclosing **book** element
  - rather than by proximity considerations
- XML allows the definition of constraints on values
  - E.g. a year must be a number of four digits

HTML vs XML: Formatting

- The HTML representation provides more than the XML representation:
  - The formatting of the document is also described
- The main use of an HTML document is to display information: it must define formatting
- XML: separation of content from display
  - same information can be displayed in different ways

HTML vs XML: Another Example

- In HTML

  <h2>Relationship force-mass</h2>
   F = M × a
- In XML

  <equation>
  <meaning>Relationship force-mass</meaning>
  <leftside> F </leftside>
  <rightside> M × a </rightside>
  </equation>

HTML vs XML: Different Use of Tags

- In both examples HTML uses same tags
- In XML completely different

- HTML tags define display: color, lists …
- XML tags not fixed: user definable tags
- XML meta markup language: language for defining markup languages

XML Vocabularies

- Web applications must agree on common vocabularies to communicate and collaborate
- Communities and business sectors are defining their specialized vocabularies
  - mathematics (MathML)
  - bioinformatics (BSML)
  - human resources (HRML)
  - …

The XML Language

An XML document consists of

- a prolog
- a number of elements

Prolog of an XML Document

The prolog consists of

- an XML declaration and
- an optional reference to external structuring documents

<?xml version="1.0" encoding="UTF-16"?>

<!DOCTYPE book SYSTEM "book.dtd">

XML Elements

- The “things” the XML document talks about
  - E.g. books, authors, publishers
- An element consists of:
  - an opening tag
  - the content
  - a closing tag

<lecturer>David Billington</lecturer>

XML Elements (2)

- Tag names can be chosen almost freely.
- The first character must be a letter, an underscore, or a colon
- No name may begin with the string “xml” in any combination of cases
  - E.g. “Xml”, “xML”

Content of XML Elements

- Content may be text, or other elements, or nothing

```
<lecturer>
    <name>David Billington</name>
    <phone> +61 – 7 – 3875 507 </phone>
</lecturer>
```

- If there is no content, then the element is called empty; it is abbreviated as follows:
  <lecturer/> for **<lecturer></lecturer>**

XML Attributes

- An empty element is not necessarily meaningless
  - It may have some properties in terms of attributes
- An attribute is a name-value pair inside the opening tag of an element

  <lecturer name="David Billington" phone="+61 − 7 − 3875 507"/>

XML Attributes: An Example

```
<order orderNo="23456" customer="John Smith"
        date="October 15, 2002">
        <item itemNo="a528" quantity="1"/>
        <item itemNo="c817" quantity="3"/>
</order>
```

The Same Example without Attributes

```
<order>
    <orderNo>23456</orderNo>
    <customer>John Smith</customer>
    <date>October 15, 2002</date>
    <item>
        <itemNo>a528</itemNo>
        <quantity>1</quantity>
    </item>
    <item>
        <itemNo>c817</itemNo>
        <quantity>3</quantity>
        </item>
</order>
```

XML Elements vs Attributes

- Attributes can be replaced by elements
- When to use elements and when attributes is a matter of taste
- But attributes **cannot** be nested

Further Components of XML Docs

- Comments
  - A piece of text that is to be ignored by parser
  - **<!-- This is a comment -->**

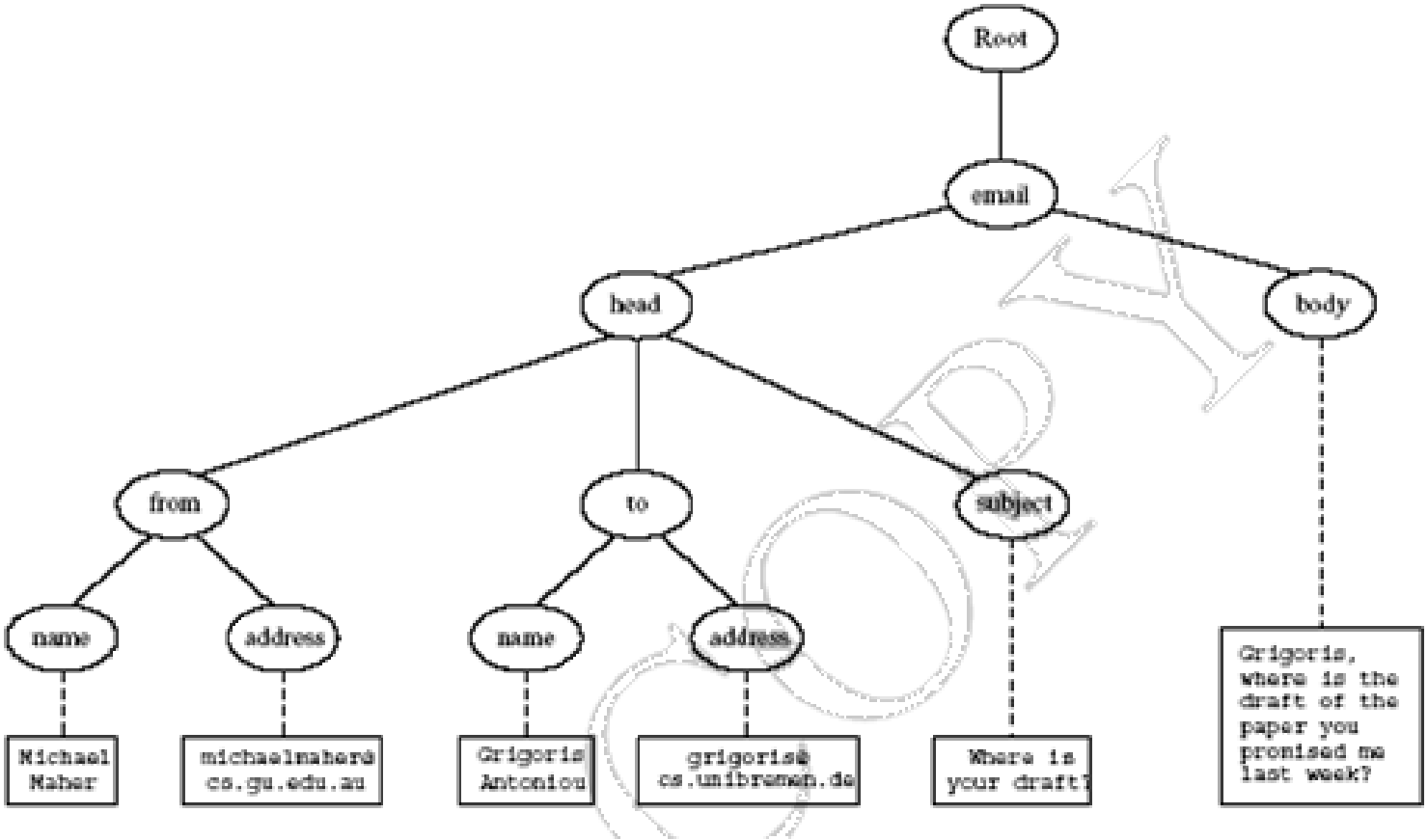
Well-Formed XML Documents

- Syntactically correct documents
- Some syntactic rules:
  - Only one outermost element (called root element)
  - Each element contains an opening and a corresponding closing tag
  - Tags may not overlap
    - **<author><name>Lee Hong</author></name>**
  - Attributes within an element have unique names

The Tree Model of XML Documents: An Example

```
<email>
    <head>
        <from name="Michael Maher"
              address="michaelmaher@cs.gu.edu.au"/>
        <to name="Grigoris Antoniou"
              address="grigoris@cs.unibremen.de"/>
        <subject>Where is your draft?</subject>
    </head>
    <body>
        Grigoris, where is the draft of the paper you promised me
        last week?
    </body>
</email>
```

The Tree Model of XML Documents: An Example (2)



XML Schema

- Define XML Structure: DTD or XML Schema
- Significantly richer language for defining the structure of XML documents
- Its syntax is based on XML itself
  - not necessary to write separate tools
- Reuse and refinement of schemas
  - Expand or delete already existent schemas
- Sophisticated set of data types, compared to DTDs (which only supports strings)

XML Schema (2)

- An XML schema is an element with an opening tag like

<schema
"http://www.w3.org/2000/10/XMLSchema"
version="1.0">

- Structure of schema elements
  - Element and attribute types using data types

Element Types

```
<element name="email"/>
<element name="head" minOccurs="1"
    maxOccurs="1"/>
<element name="to" minOccurs="1"/>
```

Cardinality constraints:

- **minOccurs="x"** (default value 1)
- **maxOccurs="x"** (default value 1)

Attribute Types

<attribute name="id" type="ID“
use="required"/>

< attribute name="speaks" type="Language"
use="default" value="en"/>

- Existence: **use="x",** where **x** may be **optional** or **required**
- Default value: **use="x" value="..."**, where **x** may be **default** or **fixed**

Data Types

- There is a variety of built-in data types
  - Numerical data types: **integer**, **Short** etc.
  - String types: **string**, **ID**, etc.
  - Date and time data types: **time**, **Month** etc.
- There are also user-defined data types
  - simple data types, which cannot use elements or attributes
  - complex data types, which can use these

Data Types (2)

- Complex data types are defined from already existing data types by defining some attributes (if any) and using:
  - **sequence**, a sequence of existing data type elements (order is important)
  - **all**, a collection of elements that must appear (order is not important)
  - **choice**, a collection of elements, of which one will be chosen

XML Schema: The Email Example

```
<element name="email" type="emailType"/>

<complexType name="emailType">
    <sequence>
        <element name="head" type="headType"/>
        <element name="body" type="bodyType"/>
    </sequence>
</complexType>
```

XML Schema: The Email Example (2)

```
<complexType name="headType">
    <sequence>
        <element name="from" type="nameAddress"/>
        <element name="to" type="nameAddress"
                 minOccurs="1" maxOccurs="unbounded"/>
        <element name="cc" type="nameAddress"
                 minOccurs="0" maxOccurs="unbounded"/>
        <element name="subject" type="string"/>
    </sequence>
</complexType>
```

XML Schema: The Email Example (3)

```
<complexType name="nameAddress">
        <attribute name="name" type="string"
        use="optional"/>
        <attribute name="address"
        type="string" use="required"/>
</complexType>
```

- Similar for **bodyType**

Namespaces

- An XML document may use more than one DTD or schema
- Since each structuring document was developed independently, name clashes may appear
- The solution is to use a different prefix for each DTD or schema
  - **prefix:name**

Addressing and Querying XML Documents

- In relational databases, parts of a database can be selected and retrieved using SQL
  - Same necessary for XML documents
  - Query languages: XQuery, XQL, XML-QL
- The central concept of XML query languages is a path expression
  - Specifies how a node or a set of nodes, in the tree representation of the XML document can be reached

XPath

- XPath is core for XML query languages
- Language for addressing parts of an XML document.
  - It operates on the tree data model of XML
  - It has a non-XML syntax

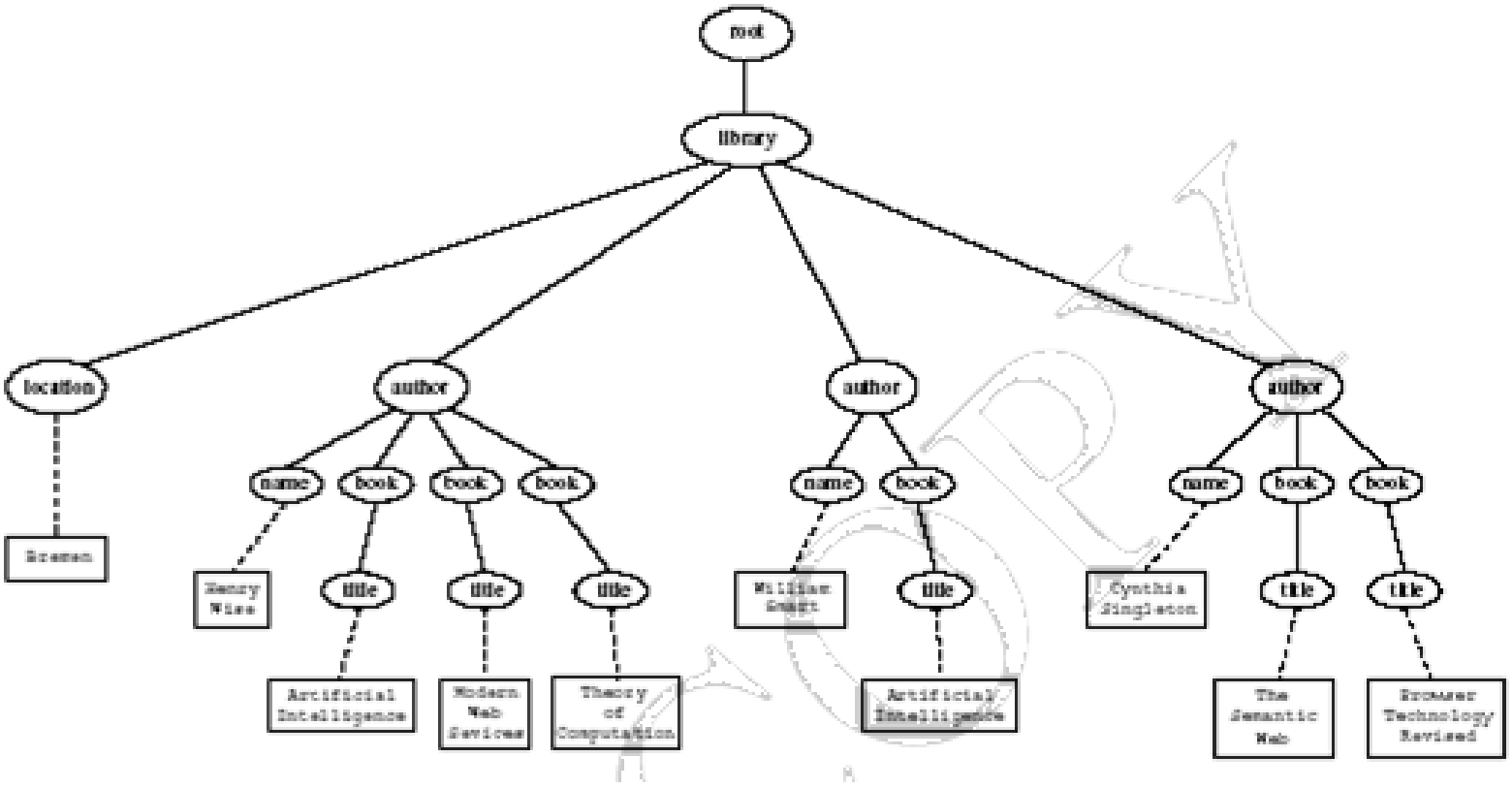
Types of Path Expressions

- **Absolute** (starting at the root of the tree)
  - Syntactically they begin with the symbol ***/***
  - It refers to the root of the document (situated one level above the root element of the document)
- **Relative** to a context node

An XML Example

```
<library location="Bremen">
    <author name="Henry Wise">
        <book title="Artificial Intelligence"/>
        <book title="Modern Web Services"/>
        <book title="Theory of Computation"/>
    </author>
    <author name="William Smart">
        <book title="Artificial Intelligence"/>
    </author>
    <author name="Cynthia Singleton">
        <book title="The Semantic Web"/>
        <book title="Browser Technology Revised"/>
    </author>
</library>
```

Tree Representation



Examples of Path Expressions in XPath

- Address all **author** elements

  /library/author

- Addresses all **author** elements that are children of the **library** element node, which resides immediately below the root
- **/t1/.../tn**, where each **ti+1** is a child node of **ti**, is a path through the tree representation

Examples of Path Expressions in XPath (2)

- Address all **author** elements

  //author

- Here ***//*** says that we should consider all elements in the document and check whether they are of type **author**
- This path expression addresses all **author** elements anywhere in the document

Examples of Path Expressions in XPath (3)

- Address the location attribute nodes within library element nodes

  /library/@location

- The symbol @ is used to denote attribute nodes

Examples of Path Expressions in XPath (4)

- Address all **title** attribute nodes within **book** elements anywhere in the document, which have the value “Artificial Intelligence”

  //book/@title="Artificial Intelligence"

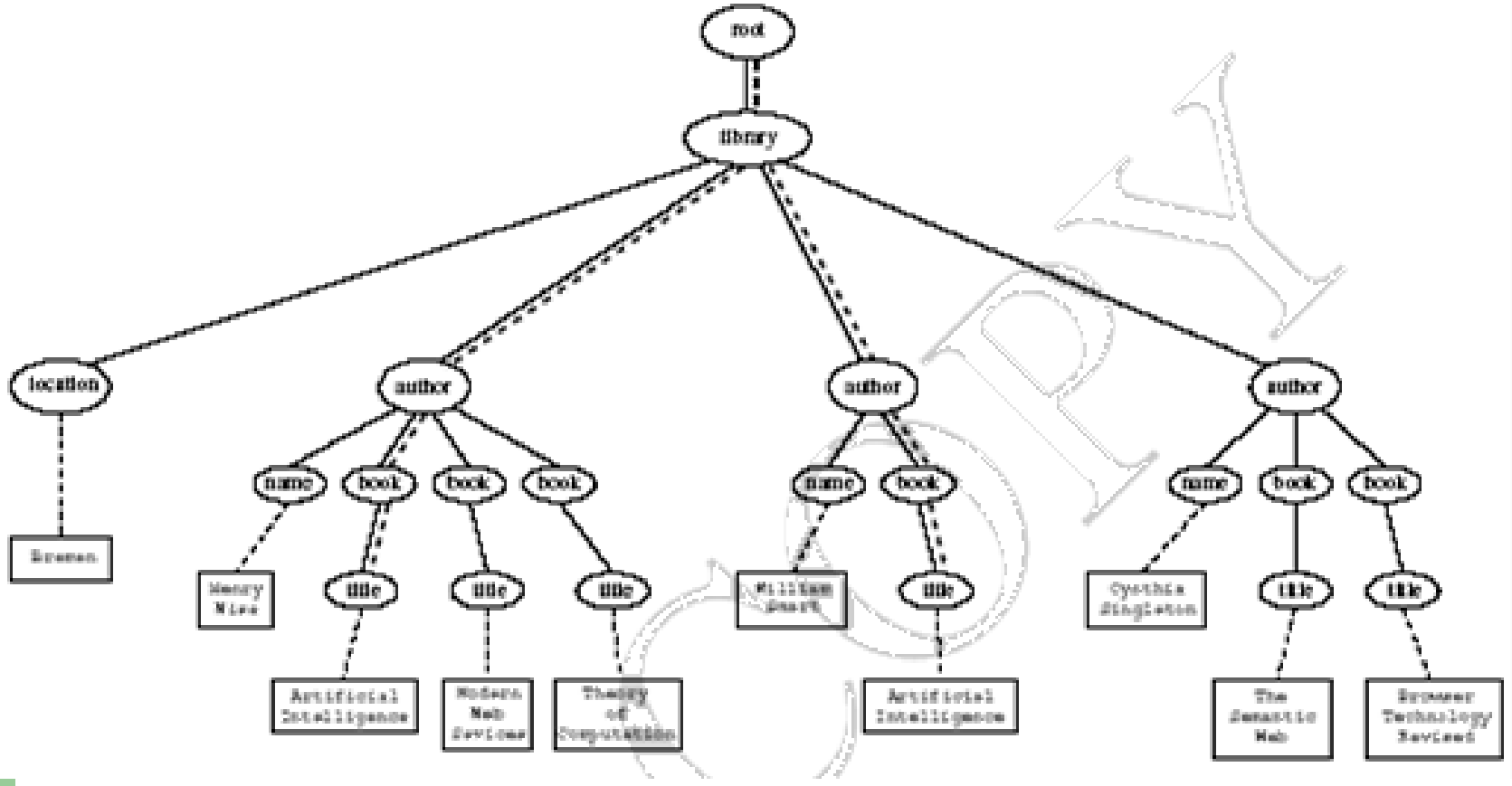
Examples of Path Expressions in XPath (5)

- Address all books with title “Artificial Intelligence”

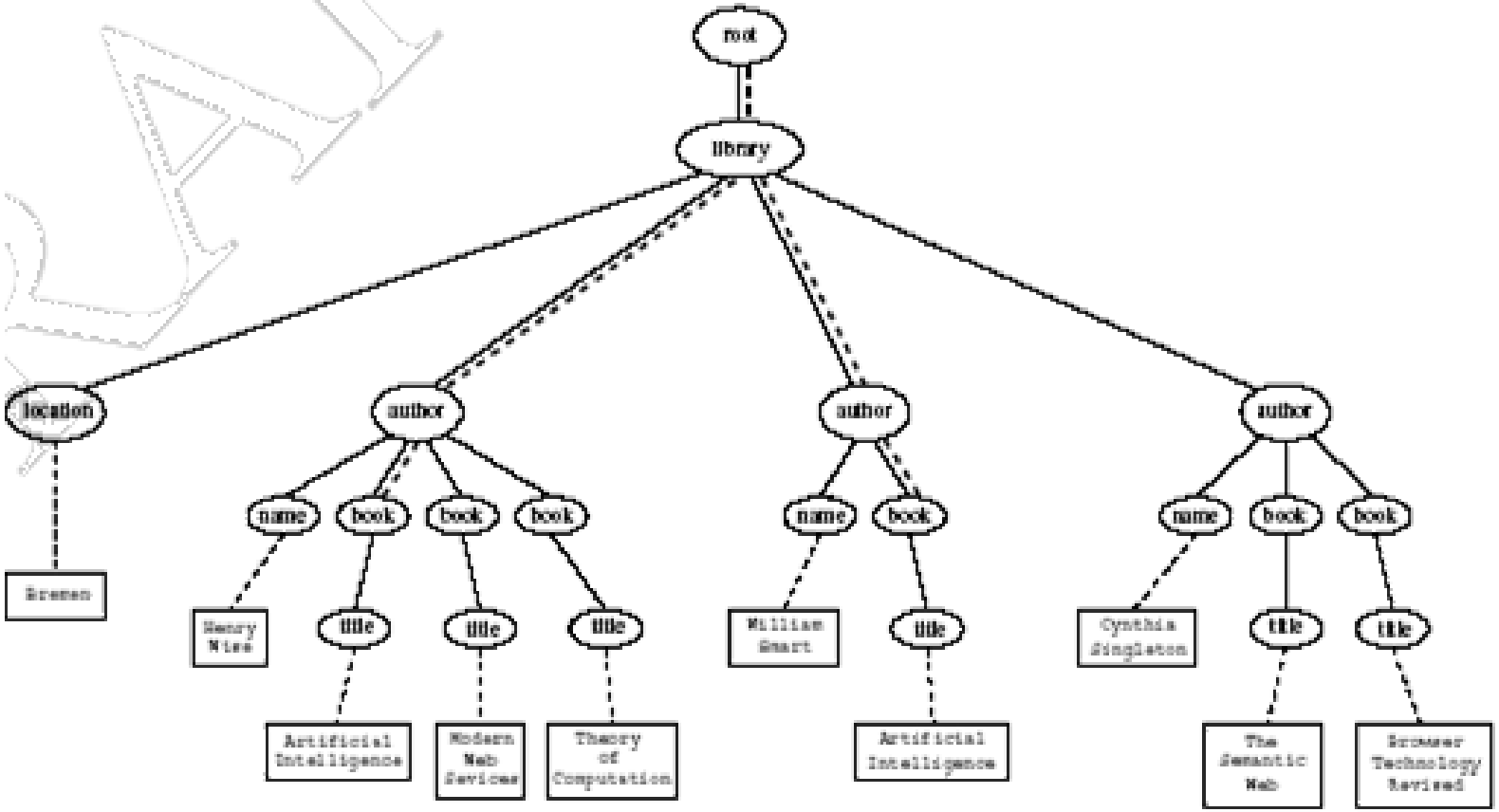
  //book[@title="Artificial Intelligence"]

- Test within square brackets: a filter expression
  - It restricts the set of addressed nodes.
- Difference with query 4.
  - Query 5 addresses **book** elements, the **title** of which satisfies a certain condition.
  - Query 4 collects **title** attribute nodes of **book** elements

Tree Representation of Query 4



Tree Representation of Query 5



Examples of Path Expressions in XPath (6)

- Address the first author element node in the XML document

  //author[1]

- Address the last book element within the first author element node in the document

  //author[1]/book[last()]

- Address all book element nodes without a title attribute

  //book[not @title]

General Form of Path Expressions

- A path expression consists of a series of steps, separated by slashes
- A step consists of
  - An **axis specifier**,
  - A **node test**, and
  - An optional **predicate**

General Form of Path Expressions (2)

- An **axis specifier** determines the tree relationship between the nodes to be addressed and the context node
  - // is such an axis specifier: descendant or self

General Form of Path Expressions (3)

- A **node test** specifies which nodes to address
  - The most common node tests are element names
  - E.g., ***** addresses all element nodes
  - **comment()** addresses all comment nodes

General Form of Path Expressions (4)

- **Predicates** (or *filter expressions*) are optional and are used to refine the set of addressed nodes
  - E.g., the expression **[1]** selects the first node
  - **[position()=last()]** selects the last node
  - **[position() mod 2 =0]** selects the even nodes
- XPath has a more complicated full syntax.
  - Here is only presented the abbreviated syntax