Methods and Tools for Management Information Systems

Lecture – Winter Semester 2006 / 2007

Dates & Obligations

- lecture: weekly tuesday, 90 minutes, 11:00 am to 1:00 pm (*cum tempere*), except october 31st; room: g22a-208
- exercise: weekly tuesday, 90 minutes, 3:00 pm to 5:00 pm (*cum tempere, unless otherwise noted*), starting on october 24th, except october 31st; room: g29-021
- news and teaching materials: http://wwwiti.cs.uni-magdeburg.de/iti_mis/
- there will be a 20-minute verbal examination at the end of the semester which you'll have to pass in order to finish the lecture successfully
- please note: you are obligated to participate in both the lecture and the exercise to pass the lecture
- contact: stefan.breitenfeld@web.de

Aims of this Lecture

- an understanding for technological basics of management information systems
- information organisation and representation on the basis of XML and related technologies
- modelling of real and real-conform circumstances using XML schemes, the resource description framework and topic maps

Content

- XML and related technologies
  - document type definitions (DTDs)
  - namespaces & entities
  - XML schema
  - XSL
- semantic technologies
  - resource description framework (RDF)
  - ontologies, web ontology language (OWL)
  - topic maps

Chapter I

XML

XML - Literature

- Young, Michael J. (2002)
  - XML. Schritt für Schritt. (Microsoft Press)
- Bongers, Frank (2004)
  - XSLT – Das umfassende Handbuch (Galileo Press)
- Van der Vlist, Eric (2003)
  - XML Schema (O'Reilly)

What is XML?

- XML (extensible markup language) is a highly flexible standard for structured, human & machine readable data streams
- it defines the syntax to create arbitrary yet similar markup languages (the so-called XML-applications)
  - example: XHTML – the XML-compatible version of HTML
- the basic idea is the strict separation between data and their representation
- it is a derived subset of SGML

Now is XML a markup language?

XML Applications

- an XML application consists of three parts
  - structure
    - defines elements, attributes, etc., which may be used in corresponding XML documents, either in form of a document type definition (DTD) or an XML scheme
  - one or several XML documents
    - contain the data
  - processing directives
    - define how the data is represented, e. g. using cascading style sheets (CSS) or the extensible stylesheet language (XSL) or the document object model (DOM)

- the official w3-specification of XML can be found here:
  - http://www.w3.org/tr/rec-xml

The Purposes of XML

- 1. XML shall be straightforwardly usable over the Internet.
- 2. XML shall support a wide variety of applications.
- 3. XML shall be compatible with SGML.
- 4. It shall be easy to write programs which process XML documents.
- 5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
- 6. XML documents should be human-legible and reasonably clear.
- 7. The XML design should be prepared quickly.
- 8. The design of XML shall be formal and concise.
- 9. XML documents shall be easy to create.
- 10. Terseness in XML markup is of minimal importance.
    - source: http://www.w3.org/tr/rec-xml

Well-formed XML Documents

- XML documents are well-formed when having an XML-conform syntax and keeping all relevant rules given in the XML specification
- an offence against any of these rules causes the XML processor to come up with a serious error message
- well-formed XML documents consist of needed and optional parts
  - needed:
    - XML declaration
    - document element (at least a root element)
  - optional:
    - comments
    - empty lines
    - processing directives

Example

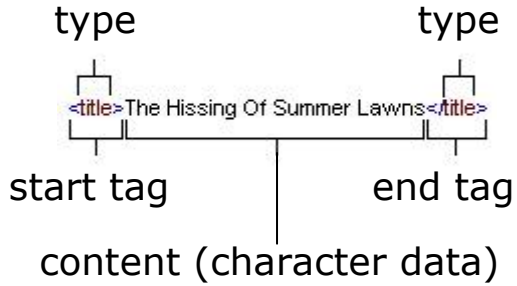
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- filename: books.xml -->

<?xml-stylesheet type="text/css" href="books.css"?>

<all_books>
  <book>
    <title>Harald Butter und der Schrein der Bleichen</title>
    <author>Johanna Krauling</author>
    <price>29,95</price>
  </book>
  <book>
    <title>Harald Butter und der Meuterelch</title>
    <author>Johanna Krauling</author>
    <price>29,95</price>
  </book>
</all_books>

<!-- Comments, empty lines and processing directives may be inserted after the document element too. -->
```

Mistakes

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- filename: books.xml -->

<?xml-stylesheet type="text/css" href="books.css"?>

<book>
    <!-- Find 5 Mistakes! -->
    <title>Harald Butter und der Schrein der Bleichen</Title>
    <author>Johanna Krauling</author>
    <price>29,95</price>
    <xml>document type</xml>
</book>
<book>
    <title><b>Harald Butter und der Meuterelch</title>
    <author>Johanna Krauling</b></author>
    < price>29,95</price>
</book>

<!-- Comments, empty lines and processing directives may be inserted after the document element too. -->
```


mls

Elements


type
type
<title>The Hissing Of Summer Lawns</title>
start tag
end tag
content (character data)

Element Types (I)

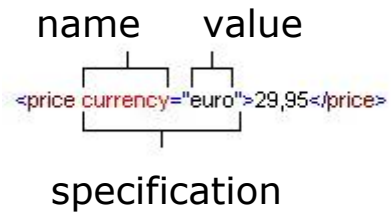
- there are no predefined element types within XML
- invalid names for element types (examples):
  - `<1st_element>…</1st_element>`
    - the first character may not be a number (only characters and underline are allowed)
  - `<element a>…</element a>`
    - space is not allowed
  - `<price_in_pounds/dollars>…</price_in_pounds/dollars>`
    - slash is not allowed
  - `<:result>…</:result>`
    - the first character may not be a double point
  - `<composer>…</Composer>`
    - start- and endtag must be written exactly the same way

Element Types (II)

- instead of `<empty></empty>`, empty elements can be written that way: `<empty/>`
- element types beginning with the prefix "xml" are reserved for further "standardisations"
  - actually, present browsers don't throw an exception, but it is recommended by the w3c not to use names beginning with "xml"
- the content of the element (the data between start- and end-tag) may not contain <, > and & (& introduces an entity reference)

Attributes (I)

- an elements start tag or an empty element may contain one or more attributes



Attributes (II)

- invalid attributes (examples):
  - `<dog source="bobtail_01.dog" source="bobtail_02.dog">`
    - the same attribute may not be used twice within one element
  - `<hitlist 1-10="top_ten.txt">`
    - the first character may not be a number (only characters and underline are allowed)
  - `<table head=""id"">`
    - use `head="'id'"` or `head='"id"'` instead
  - `<album type="<cd>">`
    - the attribute may not contain < (> is allowed within the value, but not within the attributes name)
  - `<weather forecast="cold & cloudy">`
    - & is only allowed when using a character or entity reference
  - `<setup read f i r s t !="readme.txt">`
    - the attributes name may not contain spaces

Attributes (III)

- attribute names beginning with the prefix "xml" are reserved for further "standardisations"
  - actually, present browsers don't throw an exception, but it is recommended by the w3c not to use names beginning with "xml"

<![CDATA[ ]]>

- CDATA sections start with `<![CDATA[` and end with `]]>`
- in between you can enter various characters, including <, >, & or even (X)HTML tags that are not to be interpreted by the XML processor (of course `]]>` is forbidden, as it would end the CDATA section)
- all characters are interpreted as being not markup code
- `<![CDATA[` is an XML keyword and therefore has to be written in capital letters
- CDATA sections may be inserted on every position where character data are allowed

Example

```
<?xml version="1.0" encoding="UTF-8"?>

<basics>
    <xhtml>XHTML-files are divided into <![CDATA[<html>- & <body>-Tags.]]></xhtml>
</basics>

<!-- This file is well-formed XML. -->
```

Namespaces (I)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- filename: books.xml -->

<collection>
    <item>
        <title>The Adventures Of Huckleberry Finn</title>
        <author>Mark Twain</author>
    </item>
    <item>
        <title>Madrapour</title>
        <author>Robert Merle</author>
    </item>
</collection>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- filename: cds.xml -->

<collection>
    <item>
        <title>Selling England By The Pound</title>
        <interpret>Genesis</interpret>
    </item>
    <item>
        <title>Tales From Topographic Oceans</title>
        <interpret>Yes</interpret>
    </item>
</collection>
```

Namespaces (II)

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- filename: library.xml -->

<collection
    xmlns:book="http://myhomepage.com/books"
    xmlns:cd="http://myhomepage.com/cds">

    <book:item>
        <book:title>The Adventures Of Huckleberry Finn</book:title>
        <book:author>Mark Twain</book:author>
    </book:item>
    <book:item>
        <book:title>Madrapour</book:title>
        <book:author>Robert Merle</book:author>
    </book:item>
    <cd:item>
        <cd:title>Selling England By The Pound</cd:title>
        <cd:interpret>Genesis</cd:interpret>
    </cd:item>
    <cd:item>
        <cd:title>Tales From Topographic Oceans</cd:title>
        <cd:interpret>Yes</cd:interpret>
    </cd:item>
</collection>
```

Namespaces (III)

- a namespace is declared as a special kind of attribute within the start tag of a specific element
- it separates elements with identical names but different meaning by referring to a uniform resource identifier (URI)* ** ***
- neither the XML application nor the XML processor is accessing that URI!
- an element may have two attributes with the same name – they only have to be separated by at least one namespace
- even a standard namespace can be declared, which is then valid for all undeclared elements

* URIs contain all uniform resource locators (URLs) and all uniform resource names (URNs)

** details: http://www.w3.org/adressing/

*** it *should* be a URI – it *can* be anything else (e.g. a number)

Standard Namespaces

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- filename: library.xml -->

<collection
   xmlns="http://myhomepage.com/books"
   xmlns:cd="http://myhomepage.com/cds"
   xmlns:remaster="remaster">

   <item>
      <title>The Adventures Of Huckleberry Finn</title>
      <author>Mark Twain</author>
   </item>
   <item>
      <title>Madrapour</title>
      <author>Robert Merle</author>
   </item>
   <cd:item>
      <cd:title>Selling England By The Pound</cd:title>
      <cd:interpret>Genesis</cd:interpret>
   </cd:item>
   <cd:item year="1973" remaster:year="2002">
      <cd:title>Tales From Topographic Oceans</cd:title>
      <cd:interpret>Yes</cd:interpret>
   </cd:item>
</collection>

<!-- Using the Attribute xmlns="" (an empty namespace) you can keep elements out of a standard namespace. -->
```

Testing your Skills

- Create a well-formed XML-document in iso-8859-1 containing two or three of your favourite music albums. Each album consists of its name, the artist's name and the year it was published. Comment each album with a five star rating and display it with a fictional CSS file. Add an information about its type to each album (e.g. cd, mc or lp), which must not be displayed in the browsers window. All elements within the root element (except the year) shall become part of a standard namespace.

Valid XML Documents

- an XML document is called a *valid* one, if
  - it is well-formed and
  - the prologue of the document contains a document type declaration that again contains or refers to a document type definition (DTD); the xml document corresponds to the content and structure defined within the DTD; or
  - the XML document corresponds to the content and structure of an XML schema; the latter one existing as a separate file

What are the benefits of valid XML documents?

DTD Examples

- XHTML strict DTD
  - http://www.w3.org/TR/2000/REC-xhtml1-20000126/DTD/xhtml1-strict.dtd
- XML specification DTD
  - http://www.w3.org/XML/1998/06/xmlspec-v21.dtd
- Docbook XML DTD
  - http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd

The Document Type Declaration

- the document type declaration must be inserted between the xml declaration and the root element by using the `<!DOCTYPE>` tag
- the tag `<!DOCTYPE>` is an XML keyword and therefore has to be written in capital letters
- the correct syntax is `<!DOCTYPE name [DTD]>` for internal DTDs and `<!DOCTYPE name SYSTEM "filename">` for external DTDs where *name* has to be exactly the name of the document's root element and `SYSTEM` is an XML keyword too

Example: External / Internal DTD

- files_02/external_dtd_declaration.xml
- files_02/internal_dtd_declaration.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Example for external DTD. -->
<!DOCTYPE movie SYSTEM "movie.dtd">
<!-- It doesn't matter whether  there are any comments inserted before or after the DOCTYPE element or not. -->

<movie></movie>
```

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Example for internal DTD. -->
<!DOCTYPE movie
   [
   <!ELEMENT movie ANY>
   ]
>
<!-- It doesn't matter whether  there are any comments inserted before or after the DOCTYPE element or not. -->

<movie></movie>
```

Declaring Element Types

- within the XML document only element types declared in the DTD may be used, otherwise the validation will fail
- the correct syntax is `<!ELEMENT` *`name specification`*`>` where *`name`* is the element type
- allowed specifications are
  - `EMPTY` – element may not have content
    - `<!ELEMENT` *`name`* `EMPTY>`
  - `ANY` – element may include character data and other elements without limitations or even nothing at all
    - `<!ELEMENT` *`name`* `ANY>`
  - element content
  - mixed content

Element Content I

- defined as a sequence
  - the element contains subordinated elements in a specified, comma-separated sequence; others than the specified sequence will cause a validation error
  - `<!ELEMENT name (subelement1, subelement2, subelement3)>`
- defined as a selection
  - the element may contain one of the given subordinated elements
  - `<!ELEMENT name (subelement1 | subelement2 | subelement3)>`
- modifications
  - '?' – once the previous or no element
  - '+' – one or several of the previous elements
  - '*' – none or several of the previous elements
  - in specified order: `<!ELEMENT name (sub1?, sub2+, sub3*)>`
  - in unspecified order: `<!ELEMENT name (sub1 | sub2 | sub3)+>`

Element Content II

- What does it mean?
  - `<!ELEMENT test (sub1+, sub2, sub3)>`
  - `<!ELEMENT test (sub1, sub2, sub3)?>`
- Is that useful?
  - `<!ELEMENT test (sub1 | sub2+ | sub3)>`
- Is that correct?
  - `<!ELEMENT test (sub1* | sub2 | sub3)>`

  …

  `<test />`
- Specify an element type that shall include in a certain order: sub1, sub2 and one of sub3, sub4 and sub5, where sub3 is optional.

Element Content III

- solution:
  - `<!ELEMENT test (sub1, sub2, (sub3* | sub4 | sub5))>`

Mixed Content

- only character data
  - `<!ELEMENT name (#PCDATA)>`
  - the XML keyword `#PCDATA` (parsed character data) means, that the XML processor parses the content of the specified element, looking for XML markup code; if you want to include character data that would be interpreted as being markup code, use `<![CDATA[ ]]>` sections for the element's content
- character data and subordinated elements
  - `<!ELEMENT name (#PCDATA, (sub1 | sub2)?)>`
  - this element must include character data and either the element sub1 or the element sub2 or no subordinated element

Declaring Attributes

- all attributes used within a valid XML document must be declared in the DTD
- the correct syntax is:
  - `<!ATTLIST element_name att_name att_type standard_declaration>`
- the `ATTLIST` contains all attributes for the corresponding element
  -
```
<!ATTLIST movie
    category CDATA "horror"
    year CDATA #REQUIRED>
```

Attribute Types

- type character data → `CDATA`
- type token
- type enumeration

- see the file files_02/dtd_full.xml for examples

Type Token

- `ID`
  - the attribute must have a unique identifier for each element, the first character of the value may not be a number
- `IDREF`
  - the attribute refers to another element's attribute with type `ID`
- `IDREFS`
  - same as `IDREF`, but can refer to several other elements
- `ENTITY`
  - refers to a declared external unparsed entity
- `ENTITIES`
  - can refer to several declared external unparsed entities
- `NMTOKEN`
  - name token (letters, numbers, ., -, _, : (not as first character))
- `NMTOKENS`
  - several name tokens, divided by space

Type Enumeration

- by defining name tokens
  - the value of the attribute must be one of the given words within the brackets
  - `<!ATTLIST movie category (horror | fiction | documentation) #REQUIRED>`
- by defining a `NOTATION`
  - a notation must be defined in the DTD; it describes a (file) format or identifies a program that processes a certain format
  - empty elements may not contain a `NOTATION`
  - other elements may only contain one `NOTATION`
  - `<!ATTLIST document format NOTATION (HTML | DOC | RTF) #REQUIRED>`

Standard Declarations

- `#REQUIRED`
  - the attribute's value for the corresponding element must be specified, there's no pre-set value
- `#IMPLIED`
  - the attribute's value for the corresponding element may be specified, there's no pre-set value (the value is optional)
- *`AttValue`*
  - stands for a pre-set value which is used if no value is specified by the user
- `#FIXED` *`AttValue`*
  - only a pre-set value may be specified which is used either; this makes only sense as it increases the legibility of the XML document when creating it

Declaring Namespaces

- explicit and standard namespaces for a specific element are declared the following way (files_02/dtd_namespaces.xml):

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- filename: dtd_namespaces.xml -->

<!DOCTYPE collection
  [
  <!ELEMENT collection ((item | cd:item)*)>
    <!ATTLIST collection
      xmlns CDATA #REQUIRED
      xmlns:cd CDATA #REQUIRED>
  <!ELEMENT item (title, author)>
  <!ELEMENT cd:item (cd:title, cd:interpret)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT author (#PCDATA)>
  <!ELEMENT cd:title (#PCDATA)>
  <!ELEMENT cd:interpret (#PCDATA)>
  ]
>
```

```
<collection
    xmlns="http://myhomepage.com/books"
    xmlns:cd="http://myhomepage.com/cds">

    <item>
        <title>The Adventures Of Huckleberry Finn</title>
        <author>Mark Twain</author>
    </item>
    <cd:item>
        <cd:title>Selling England By The Pound</cd:title>
        <cd:interpret>Genesis</cd:interpret>
    </cd:item>
</collection>
```

Combining DTDs

- it is possible to combine external and internal DTDs
- if there exists an element, attribute, entity or notation which is declared in both internal and external DTD under the same name, only the internal declaration is used
- the correct syntax is:
  - `<!DOCTYPE name SYSTEM "filename.dtd"`

    `[`

    `<!ELEMENT test (#PCDATA)>`

    `…`

    `]`

    `>`

<![IGNORE[and <![INCLUDE[

- for deactivating a block with markup code temporarly (e.g. when developing) use the XML keyword `<![IGNORE[ ]]>`
- for activating a block with markup code temporarly (e.g. when developing) use the XML keyword `<![INCLUDE[ ]]>`

Entities

- according to the XML specification, the term "entity" is used for the following circumstances:
  - the whole XML document itself (a so-called document entity)
  - an external DTD
  - an external file which is declared as an external entity within the DTD; the XML document contains a reference to the external file
  - a string, surrounded by quotation marks which is declared as an internal entity within the DTD; the XML document contains a reference to the string
- to reduce the size of XML documents
- to modularize XML documents
- to integrate different types of data

Using Entities

- files_03/entities_example.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- filename: entities_example.xml -->

<!DOCTYPE article
[
   <!ELEMENT article (frontpage, introduction, content+)>
   <!ELEMENT frontpage (#PCDATA)>
   <!ELEMENT introduction (#PCDATA)>
   <!ELEMENT content (#PCDATA)>

   <!ENTITY themes SYSTEM "themes.xml">
   <!ENTITY title "A short history of XML">
]>

<article>
   <frontpage>Title: &title;</frontpage> <!-- &title; refers to the entity "title" -->
   <introduction>This article is about: &themes;</introduction> <!-- &themes; refers to the entity "themes" -> the file "themes.xml" is loaded and included here -->
   <content>content</content>
</article>
```

Types Of Entities (I)

- general vs. parameter entities

- general entities
  - contain XML text or other text used in the XML document (within the root element)
- parameter entities
  - contain XML text (markup code) used within the DTD

Types Of Entities (II)

- internal vs. external entities

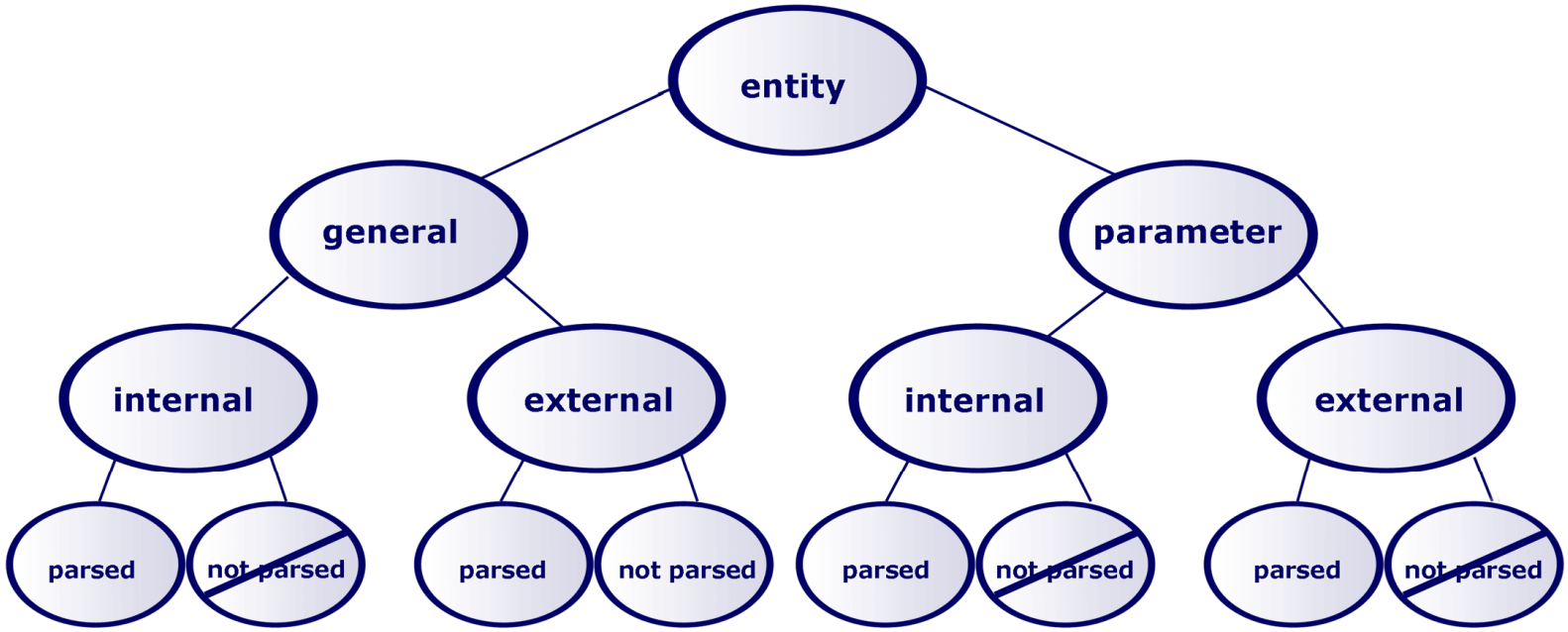
- internal entities
  - strings, surrounded in quotation marks
- external entities
  - are saved in separate files

Types Of Entities (III)

- parsed vs. unparsed entities

- parsed entities
  - contain XML text; the reference is replaced by the entity's content which then is parsed by the XML processor
- unparsed entities
  - contain XML data, text or other data (e.g. images); the XML processor does not access the external file but passes it on to the application that processes the XML document

Types Of Entities (IV)



Entity Names And Values

- names
  - names may only start with letters or an underline, not with a number
  - names may not start with "xml"
  - the XML processor is case sensitive
- values
  - values must be surrounded by apostrophes or quotation marks
  - values may not contain the surrounding character
  - values may not contain the characters & and % (exceptions can be found in chapter 4 of the XML specification, see http://www.w3.org/tr/rec-xml)
  - values must be valid in the context where they are to be inserted

Internal Parsed General Entities

- declaration
  - `<!ENTITY name "value">`
- example
  - `<!ENTITY abb "(This is an abbreviation)">`

```
<content>PHP &abb;</content>
<content>HTML &abb;</content>
```

External Parsed General Entities

- declaration
  - `<!ENTITY name SYSTEM "filename"`
- example
  - `<!ENTITY themes SYSTEM "http://www.myhomepage.com/themes.xml">`

```
<description>You'll find all themes here: &themes;</description>
```

External Unparsed General Entities

- declaration
  - `<!ENTITY name SYSTEM "filename" NDATA notation_name>`
- example
  - `<!ENTITY cover SYSTEM "cover.gif" NDATA GIF>`

```
<!ELEMENT cover_image EMPTY>
<!ATTLIST cover_image source ENTITY #REQUIRED>


<cover_image source="cover" />
```

- the keyword `NDATA` indicates that the file contains unparsed data

Internal Parsed Parameter Entities

- declaration
  - `<!ENTITY % name "value">`
- example
  -
```
<!ENTITY author "
    <!-- the author's information -->
    <!ELEMENT author (name, birthdate)>
    <!ELEMENT name (#PCDATA)>
    <!ELEMENT birthdate (#PCDATA)>">
```
- the value must include at least one complete markup declaration, corresponding to the DTD where it is used

External Parsed Parameter Entities

- declaration
  - `<!ENTITY % name SYSTEM "filename">`
- example
  -
```
<!DOCTYPE library [
  <!ELEMENT library (book | cd)*>
  <!ENTITY % book SYSTEM "book.dtd">
  <!ENTITY % cd SYSTEM "cd.dtd">

  %book;
  %cd;
]>
```

Inserting Entity References

- general entities:
  - `&name;`
- parameter entities:
  - `%name;`
- each entity must be declared before it is referred

Notations

- declaration
  - `<!NOTATION name SYSTEM "system_literal">`
- example
  - `<!NOTATION html SYSTEM "c:\program_files\mozilla\firefox.exe">`

- notations are always used with external unparsed general entities
- the system literal is not accessed by the XML processor, it is only forwarded to the application that processes the XML document (i.e. a webpage script)
- the system literal describes the format (normally either by using a URI referring to more information or by specifying a program to process the format)

Inserting Character References

- to insert characters or letters that can not be found on the keyboard, you can use decimal or hexadecimal codes corresponding to the ISO/IEC-10646 code (see http://www.iso.org/iso/en/ISOOnline.frontpage for details)
- decimal for "A": A
- hexadecimal for "A": A
- more:
  - http://unicode.e-workers.de/entities.php
  - http://www.htmlhelp.com/reference/html40/entities/

Predefined Entities (excerpt)

&	&#38;	&amp;
<	&#60;	&lt;
>	&#62;	&gt;
'	&#39;	&apos;
"	&#34;	&quot;

XML Schema

- using an XML schema is the second possibility to create valid XML documents (see the chapter about DTDs for the first one)
- XML schema files have two advantages compared to DTDs:
  - they don't have an own type of notation but are written in the typical XML syntax
  - they offer more details and functions to describe the content and the structure of a class of documents

    -> but the number of functions can make the creation of a schema file a real challenge
- this lecture will only provide you with a basic introduction due to the complexity of this topic, for more detailed informations use the corresponding literature
- an XML schema is always stored in a separate file

Resources

- the w3 specification can be found here:
  - part 0 (primer): http://www.w3.org/TR/xmlschema-0/
  - part 1 (structures): http://www.w3.org/TR/xmlschema-1/
  - part 2 (data types): http://www.w3.org/TR/xmlschema-2/
- for validating a document using XML schema you need the MSXML package on your machine (windows users only); the last version (MSXML 6.0) can be found here:
  - http://www.microsoft.com/downloads/details.aspx?FamilyID=993C0BCF-3BCF-4009-BE21-27E85E1857B1&displaylang=en
- for more resources, tools and examples, look here:
  - http://www.w3.org/XML/Schema#resources

Basics

- schema files own the file extension *.xsd (abbreviated for "XML schema definition language")
- an XML document which is valid against a schema file is called an instance document (or simply an instance) of that schema
- the basic structure of a schema file is as follows:
  - `<?xml version="1.0"?>`

    `<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema> *`

    `< . . . >`

    `</xsd:schema>`

* some applications use `xmlns:xs=http://www.w3.org/2001/XMLSchema`

Declaring Elements

- elements may be declared with:
  - simple, pre-defined data types
  - simple, self-defined data types
  - complex data types
  - element content
  - mixed content
  - no content

Simple, Pre-Defined Data Types

- declaration
  - `<xsd:element name="`*name*`" type="xsd:`*type*`" minOccurs="" maxOccurs=""/>`
- example
  - `<xsd:element name="book" type="xsd:string" minOccurs="0"/>`
- `minOccurs` and `maxOccurs` define how often an element may occur (corresponding to `?`, `+` and `*` in DTDs); not defined means "1"; `minOccurs` must be smaller than `maxOccurs`; the latter one can be set to `unbounded`; the declaration of elements that may only occur exactly once may not contain `minOccurs` and `maxOccurs`
- there are lots of pre-defined data types

Pre-Defined Data Types (I)

Data Type	Description	Examples
`xsd:string`	string	This is a string.
`xsd:boolean`	true/false or 1/0	true
`xsd:decimal`	integer or decimal number	-5.2; -3; 726; 2,6
`xsd:integer`	integer	-389; 0; 35
`xsd:positiveInteger`	positive integer, without 0	35
`xsd:negativeInteger`	negative integer, without 0	-389
`xsd:date`	date (yyyy-mm-dd)	2006-11-21

Pre-Defined Data Types (II)

Data Type	Description	Example
xsd:time	time (hh:mm:ss.ss)	11:30:00.00 or 11:30:00
xsd:dateTime	date & time (yyyy-mm-ddThh:mm:ss.ss)	2006-11-21T11:30:00.00
xsd:gMonth	month, gregorian calendar (mm)	11
xsd:gYear	year, gregorian calendar (yyyy)	2006
xsd:gDay	day, gregorian calendar (dd)	21
xsd:gYearMonth	year & month, gregorian calendar (yyyy-mm)	2006-11
xsd:anyURI	a uniform resource identifier	urn:loc.gov:books

Simple, Self-Defined Data Types

- declaration
  - <xsd:element name="*name*">
    <xsd:restriction base="*pre-defined_type*">
    <. . . restrictions . . .>
    </xsd:restriction>
    </xsd:element>

Restrictions (I)

- minimum / maximum value
- declaration
  -
```
<xsd:restriction base="xsd:decimal">
  <xsd:minExclusive value="value"/>
  <xsd:maxExclusive value="value"/>
</xsd:restriction>
```

Restrictions (II)

- enumerations
- declaration
  -
```
<xsd:restriction base="xsd:string">
  <xsd:enumeration value="string1"/>
  <xsd:enumeration value="string2"/>
  <xsd:enumeration value="string3"/>
</xsd:restriction>
```

Restrictions (III)

- pattern
- declaration
  - ```
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="\d{1}-\d{4}-\d{4}-\d{1}"/>
    </xsd:restriction>
    ```
- example
  - ```
    <ISBN>0-7356-1020-7</ISBN>
    ```

Designated Data Types

- example
  -
```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
  <xsd:simpleType name="ISBNtype">
  <xsd:restriction base="xsd:string"/>
  . . .
  </xsd:restriction>
  </xsd:simpleType>
  . . .
  <xsd:element name="ISBN" type="ISBNtype"/>
</xsd:schema>
```

Complex Data Types

- the element may contain character data, elements and attributes
- declaration
  - `<xsd:element name="name" type="xsd:anyType"/>`

Element Content: `sequence`

- declaration
  - ```
    <xsd:element name="name">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="name" type="type"> *
          <xsd:element name="name" type="type">
          <xsd:element name="name" type="type">
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    ```
- you may alternatively use `<xsd:element ref="name"/>` which means a reference to the corresponding element and its type definition
    ```

Element Content: `choice`

- declaration
  - `<xsd:element name="`*name*`">`

```
<xsd:complexType>
        <xsd:choice>
                <xsd:element name="name" type="type">
                <xsd:element name="name" type="type">
                <xsd:element name="name" type="type">
        </xsd:choice>
</xsd:complexType>
</xsd:element>
```

Mixed Content

- declaration
  -
```
<xsd:element name="name">
  <xsd:complexType mixed="true">
    <xsd:sequence>
      <xsd:element name="name" type="type">
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```
- -> the outer element may contain character data before or after the inner element

No Content

- declaration 1
  - <xsd:element name="*name*">
    <xsd:complexType>
    </xsd:complexType>
    </xsd:element>
- declaration 2
  - <xsd:element name="*name*" abstract="true"/>

Declaring Attributes

- declaration 1
  - `<xsd:attribute name="name" type="pre-defined_type" use="value"/>`
- declaration 2
  - `<xsd:attribute name="name" use="value">`

```
<xsd:simpleType>
        <xsd:restriction base="pre-defined_type">
        < . . . >
        </xsd:restriction
</xsd:simpleType>
</xsd:attribute>
```

- data types and restrictions are the same as with elements

Values For `use`

- `use="optional"` (or no `use` at all)
  - the attribute is optional
- `use="required"`
  - the attribute is required within the document
- `use="prohibited"`
  - the attribute is not allowed to be specified
- `fixed="`*`value`*`"`
  - same as #FIXED in DTDs, no other value will be allowed
- `default="`*`value`*`"`
  - the default value will be used if no other value is specified

Adding Attributes (I)

- the attribute declaration is integrated into the corresponding `<xsd:element></xsd:element>` tag
- it may not occur *before* an `<xsd:sequence>` or an `<xsd:choice>` tag
- for an example, please look at files_04/xsd_full.xsd

Adding Attributes (II)

- declaration for adding an attribute to an empty element
  - ```
    <xsd:element name="name">
      <xsd:complexType>
              <xsd:attribute name="name" type="pre-defined_type"/>
      </xsd:complexType>
    </xsd:element>
    ```
- declaration for adding an attribute to an element containing only character data
  - ```
    <xsd:element name="name">
      <xsd:complexType mixed="true">
              <xsd:attribute name="name" type="pre-defined_type"/>
      </xsd:complexType>
    </xsd:element>
    ```

Attribute References

- alternatively you can use so-called attribute groups to specify attributes outside the corresponding element
- declaration
  - ```
    <xs:element name="name">
      <xs:complexType>
              <xs:attributeGroup ref="attlist.name"/>
      </xs:complexType>
    </xs:element>

    <xs:attributeGroup name="attlist.name">
      <xs:attribute name="name" use="value"/>
    </xs:attributeGroup>
    ```
    ```

Special Attribute Types

- ID
  - `type="xsd:ID"`
- IDREF
  - `type="xsd:IDREF"`
- entity
  - `type="xsd:ENTITY"`
- token
  - `type="xsd:token"`
- token used for an enumeration
  -
```
<xsd:simpleType>
  <xsd:restriction base="xs:token">
            <xsd:enumeration value="value1"/>
            <xsd:enumeration value="value2"/>
  </xsd:restriction>
```

Notations

- notations cannot be used directly in a schema file
- for details please look at:
  - http://www.w3.org/TR/xmlschema-2/#NOTATION

Including Other Schema Files

- declaration (after the `<xsd:schema>` element)
  - `<xs:include schemaLocation="`*`path_to_file`*`"/>`

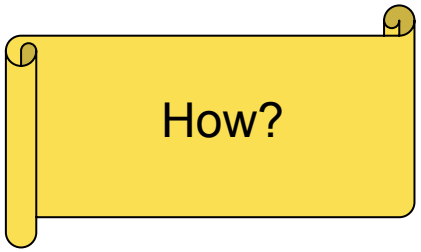
Please note: Many programs offer the possibility to convert an existing and valid DTD into an XML schema file (but without converting notations)!

Testing Your Skills

- Specify an element type that shall include in a certain order: sub1, sub2 and one of sub3, sub4 and sub5, where sub3 is optional. Plus, the element contains the attribute att1, which is of type `positiveInteger` with a maximum value of 20.

Displaying XML Documents

- XML documents can be displayed within a browser
  - using Cascading Stylesheets (CSS)
  - using data binding with the Data Source Object (DSO)
  - using the Document Object Model (DOM)
  - using Extensible Stylesheet Language Transformations (XSLT)
- XML documents can be displayed within a certain program
  - Java, C#, PHP?



A – Cascading Style Sheets

- are the most simple way to display XML documents
  - most web designers know how to create and work with CSS files
  - current web browsers support CSS to a high degree
- are less flexible than other technologies
  - the content of the elements cannot be modified or re-structured
  - attributes, unparsed entities and processing directives cannot be accessed

- CSS instructions can be stored in separate stylesheet files or be part of the XML document itself

Using Stylesheets

- including separate CSS files:
  - `<?xml-stylesheet type="text/css" href="`*filename*`"?>`
- CSS instructions within the XML file:
  - `<`*element_name* `STYLE='`*instruction1; instruction2;*`'>`
    content`</element_name>`

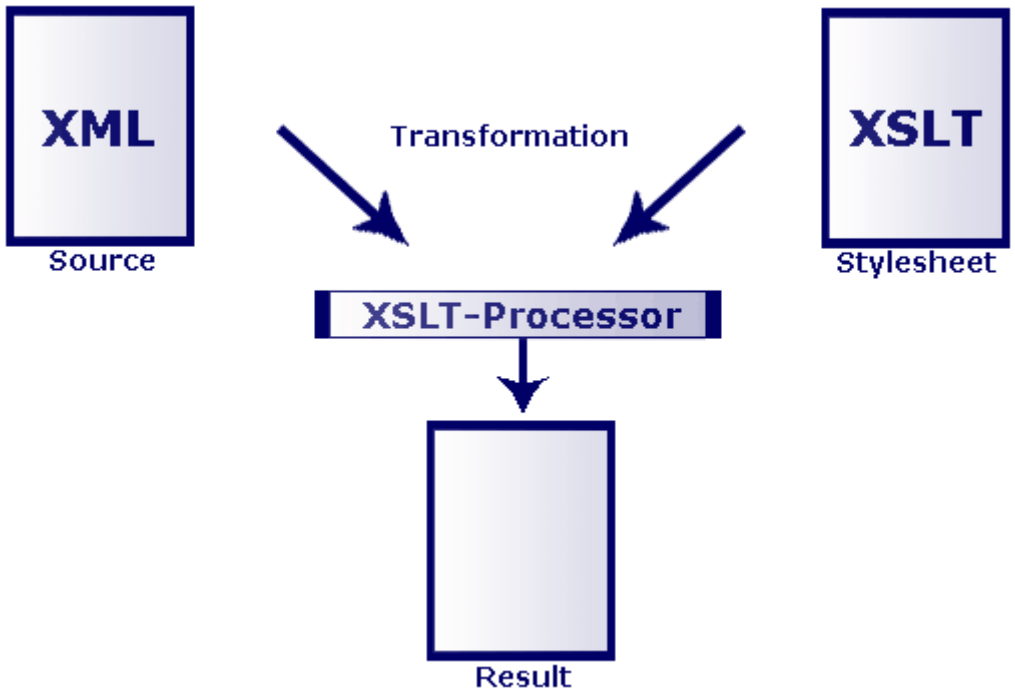
Resources

- Herold, H. (2002): Das HTML / XHTML – Buch. Suse Press.
- Laborenz, K. (2006): CSS-Praxis. Galileo Press.
- Lubkowitz, M. (2006): Webseiten programmieren und gestalten. Galileo Press.
- Shafer, D. / Yank, K. (2006): Cascading Stylesheets. Dpunkt Publishing.
- CSS specifications:
  - http://www.w3.org/tr/rec-css1
  - http://www.w3.org/tr/rec-css2

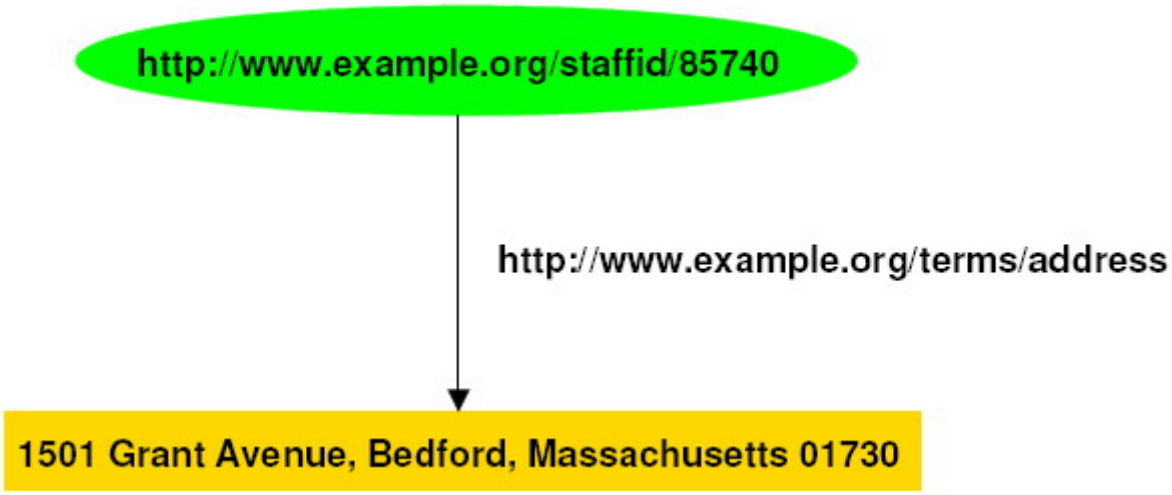
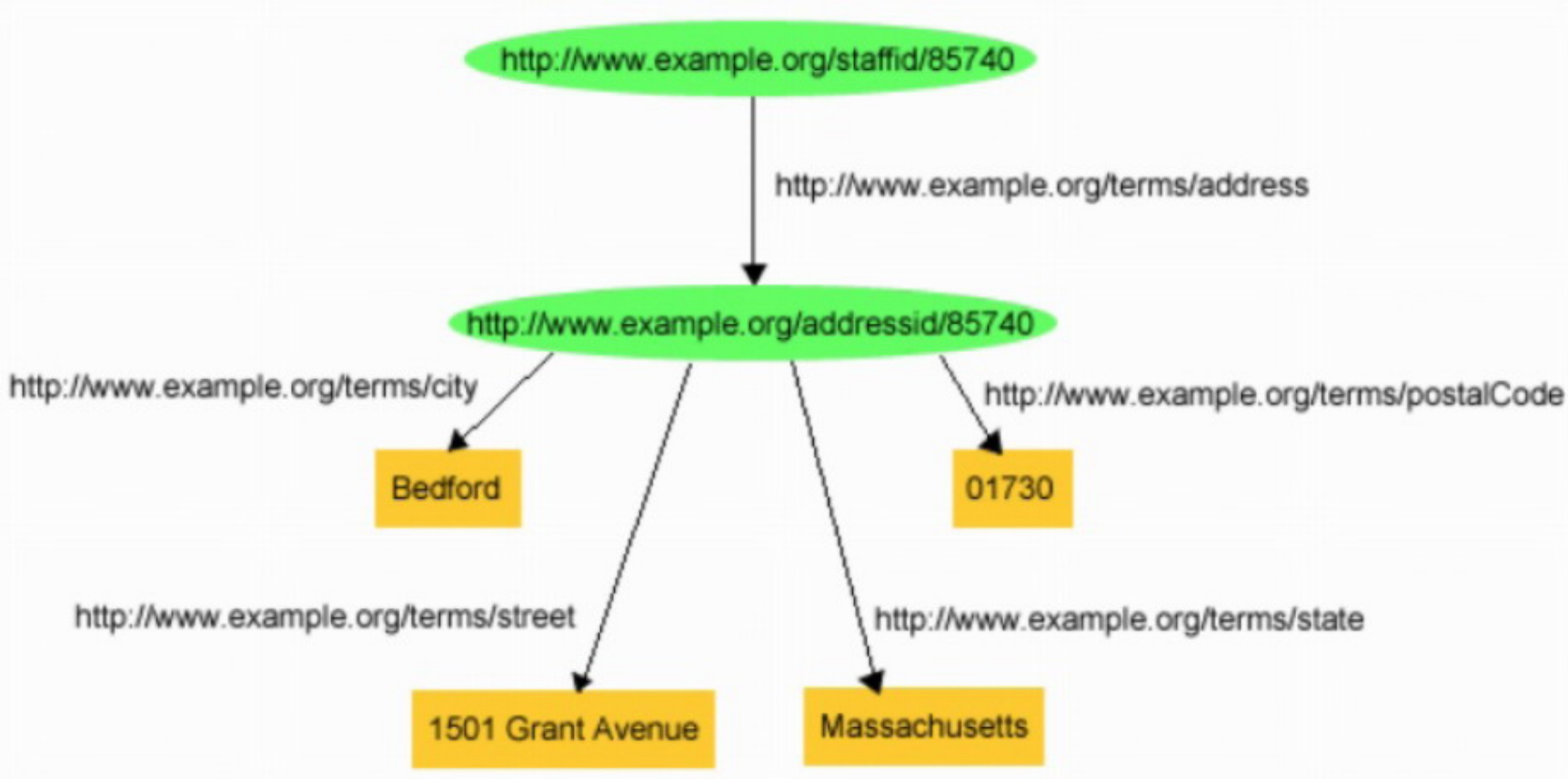
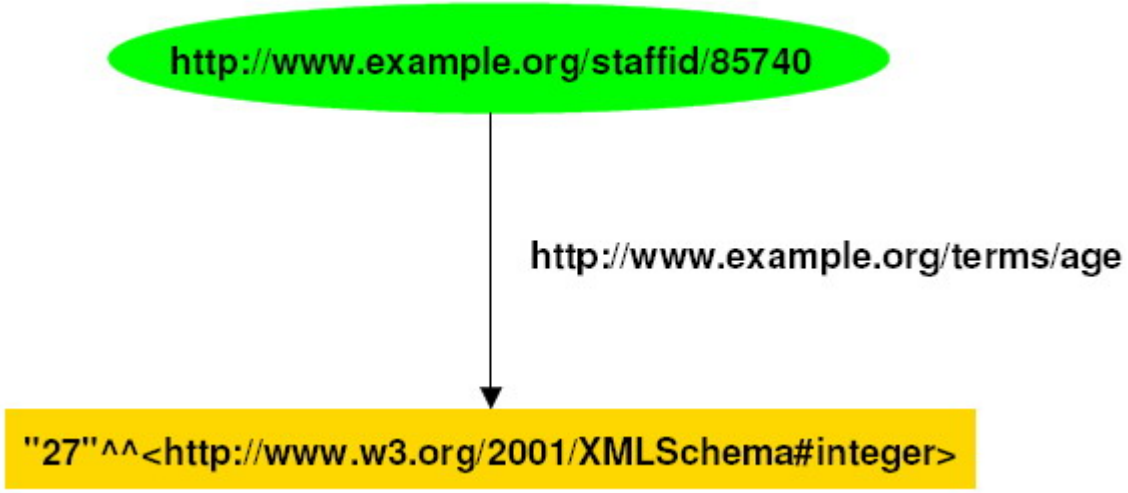
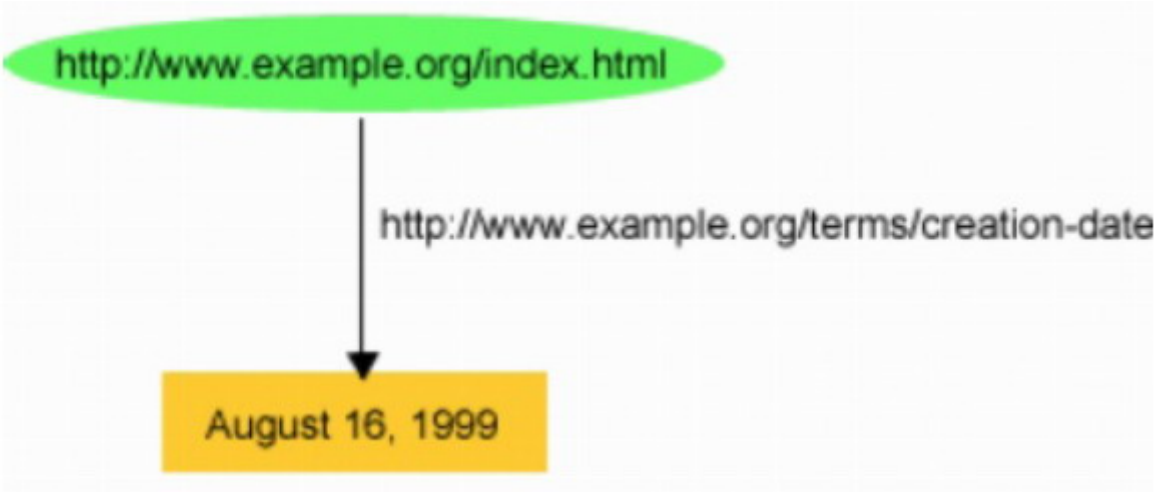
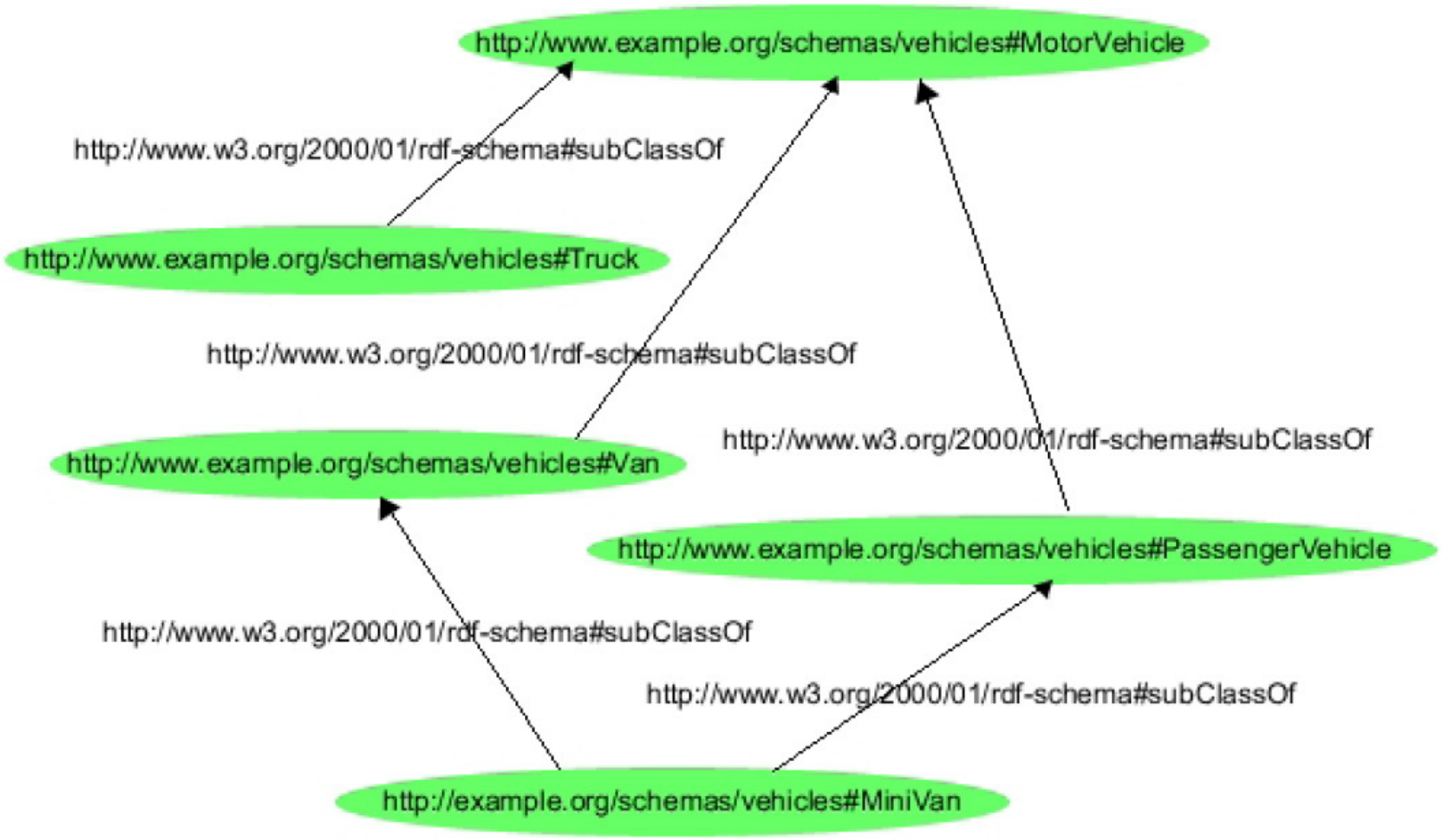
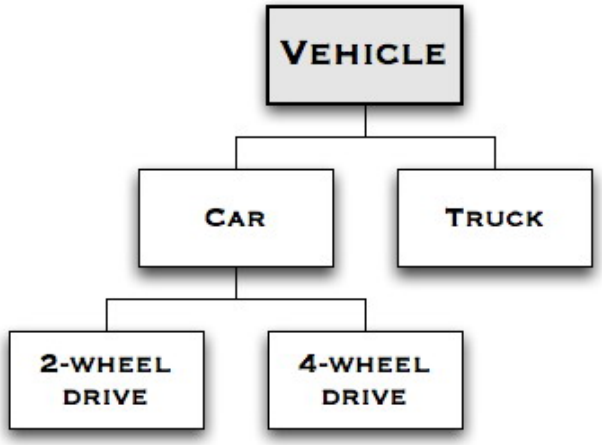
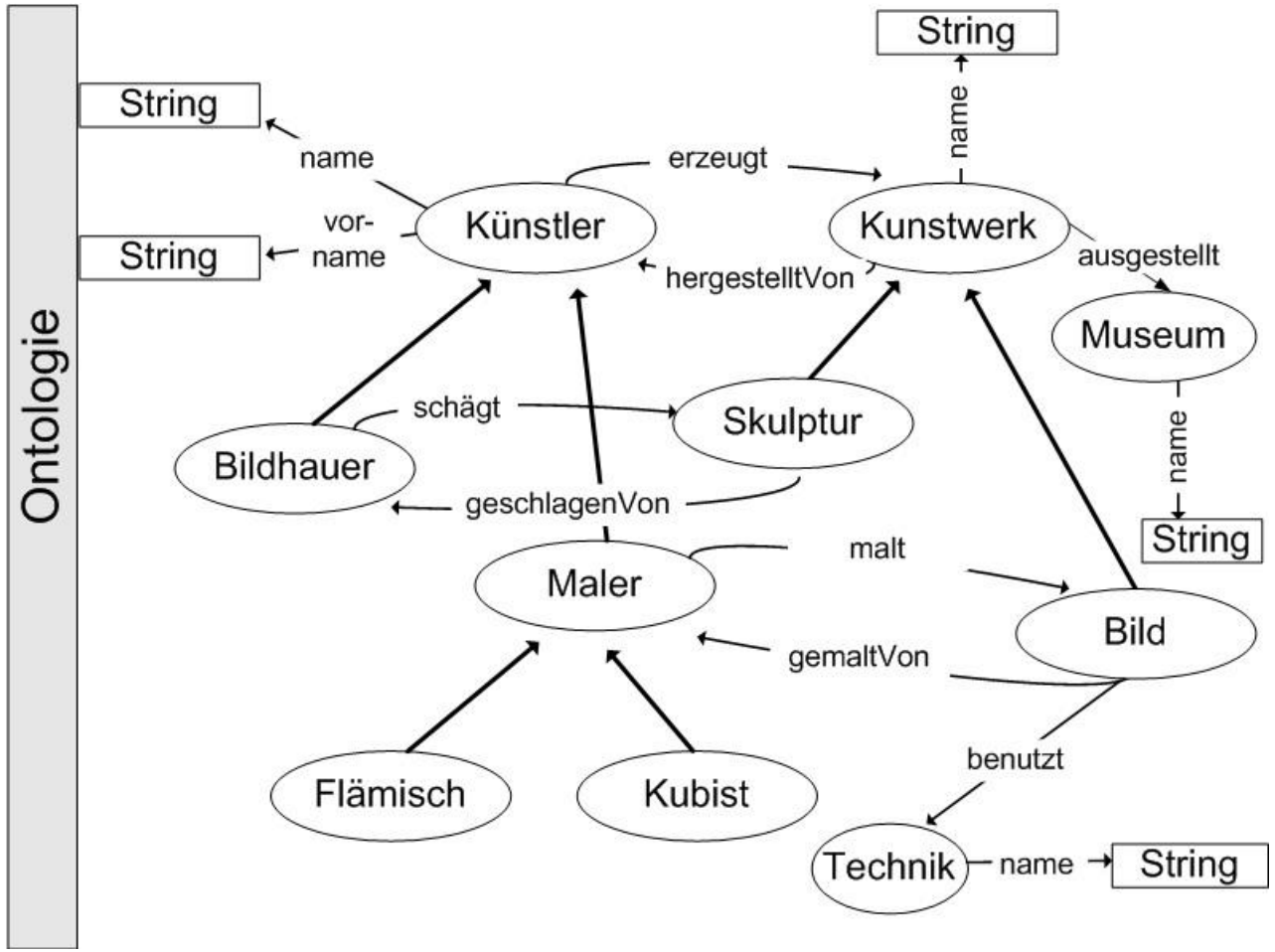
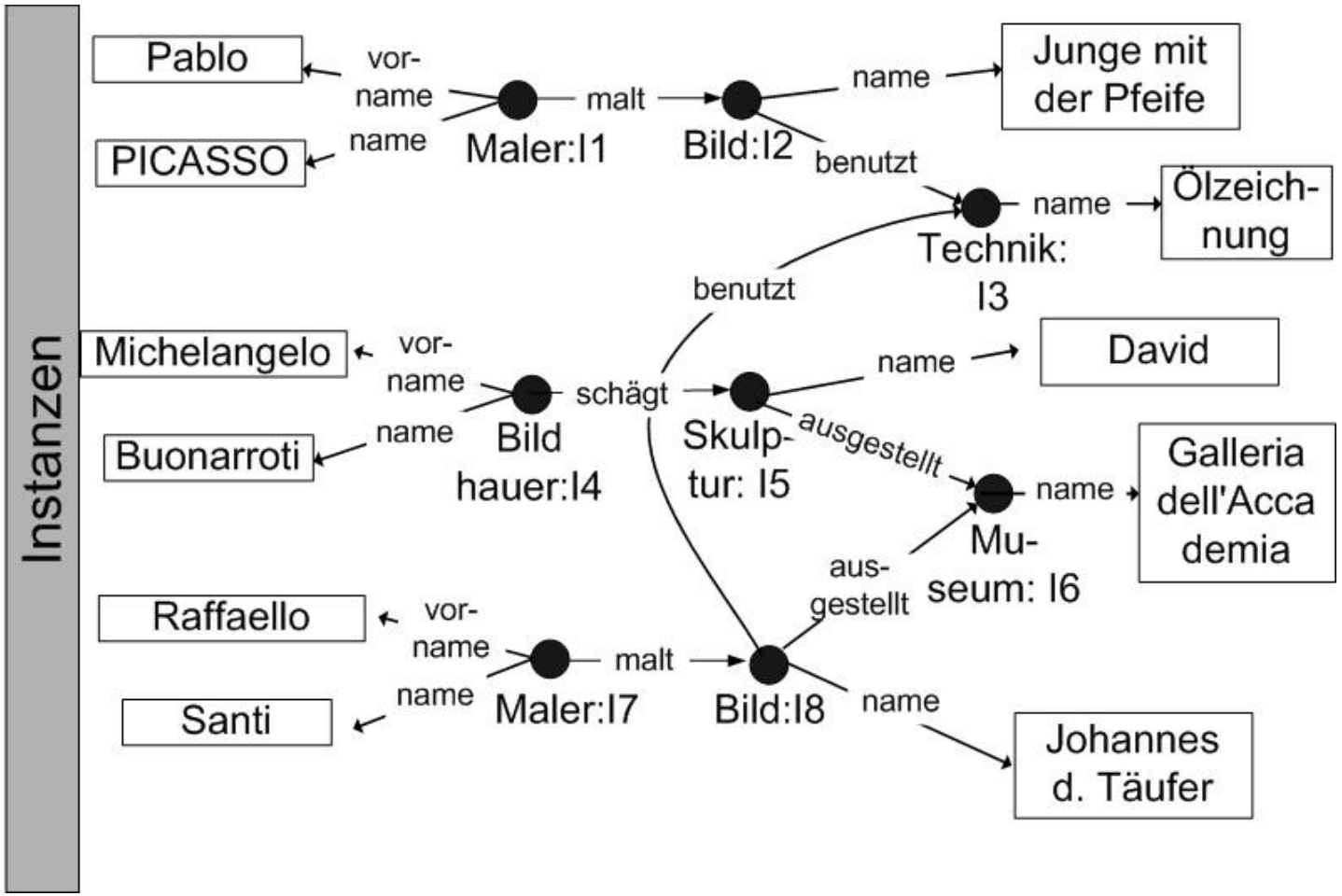
B – Data Binding

- XML documents are integrated with (X)HTML documents and (X)HTML standard elements (like <SPAN> or <TABLE> are bound to XML elements
- the (X)HTML elements then display the content of the corresponding XML elements
- data binding can only be used with symmetrical structured XML documents (using a DTD to ensure the file has the required structure is strongly recommended)
  - that means: a document has a root element which contains a certain amount of elements of the same type (the data sets) which in turn contain a certain amount of elements with character data (the fields) – i.e. a library of books with each book consisting of an author and a title
- data binding is not supported by all current browsers
  - MS Internet Explorer / Mozilla Firefox: partly different syntax and functionality

Integrating The Documents (I)

- XML and HTML documents are integrated via so-called data islands
- example for a fictional file "library.xml":
  - ```
    <html>
     <head>
      <title> … </title>
     </head>
     <body>
      <xml id="dsolibrary" src="library.xml"></xml>
      <!-- more HTML elements -->
     </body>
    </html>
    ```
- the browser checks the well-formedness / validity of the XML file
    ```

Integrating The Documents (II)

- it is also possible to include the whole XML root element with its content in the HTML file, but
  - this is not quite the XML philosophy, as data and their representation are no longer separated
  - maintenance of the XML document is more lavish, especially if the XML document is displayed in more than one HTML documents

Binding The Elements

- there are two binding methods:
  - table data binding – a <TABLE> element is bound to the XML data and displays all data sets of the XML document
  - data set binding – non-table-elements, e.g. <SPAN>, are bound to XML elements displaying only one element any one time

Table Data Binding (I)

- example: library.xml
  -
```
<body>
 <xml id="dsolibrary" src="library.xml"></xml>
 <table datasrc="#dsolibrary">
  <thead> … </thead>
  <tr>
   <td><span datafld="element_name1"></span></td>
   <td><span datafld="element_name2"></span></td>
   …
  </tr>
 </table>
</body>
```

Table Data Binding (II)

- although only one table row is defined, the browser repeats that row for all available data sets from the XML document
- <SPAN> is used because <TD> is not able to bind XML elements

Grouping Data Sets

- data sets can be grouped in order to create several pages from one XML document
- an id must be specified
- the attribute `datapagesize` defines the number of data sets per page
  - `<table id="books" datasrc="#dsolibrary" datapagesize="6">`
- predefined methods for navigation can be implemented (e.g. by using buttons)
  - `<button onclick="books.nextPage()">next page</button>`
  - if the corresponding page does not exist, clicking the button is ignored by the browser

Navigation Methods

method	result	example
firstPage	shows the first page with data sets	`books.firstPage()`
previousPage	shows the previous page with data sets	`books.previousPage()`
nextPage	shows the next page with data sets	`books.nextPage()`
lastPage	shows the last page with data sets	`books.lastPage`

Data Set Binding

- only one data set is displayed
  -
```
<body>
 <xml id="dsolibrary" src="library.xml"></xml>
 <span datasrc="#dsolibrary" datafld="element_name1"></span>
 <br />
 <span datasrc="#dsolibrary" datafld="element_name2"></span>
</body>
```
- attributes can be bound the same way, just replace *element_name* by attribute_name
  - if the attribute's name matches the name of another element, use !*element_name* and attribute_name instead
- the data source object provides methods to search through the data sets

Search Methods

method	current data set changes to	example
moveFirst	the document's first data set	`dsolibrary.recordset.moveFirst()`
movePrevious	the previous data set	`dsolibrary.recordset.movePrevious()`
moveNext	the next data set	`dsolibrary.recordset.moveNext()`
moveLast	the document's last data set	`dsolibrary.recordset.moveLast()`
move	the data set with the specified number	`dsolibrary.recordset.move(5)`

Resources

- for a list of all binding-capable HTML elements see:
  - Young, M. (2002): XML – Schritt für Schritt. Microsoft Press. (p.339ff.)

C – Document Object Model

- the DOM works with all kinds of well-formed XML documents, it doesn't expect a certain structure
- all components are accessible (even attributes, processing directives, notations and comments) – they are represented by a structured set of nodes
- implementations:
  - client side: MS Internet Explorer / Mozilla Firefox with partly different approaches, syntax and functionality
  - server side: e.g. PHP5

Node Types (I)

Node Type	XML component	nodeName	nodeValue
document	root element	*#document*	*null*
element	element	element name	*null*
text	text within the super-ordinated component	*#text*	text of the super-ordinated component
attribute	attribute	attribute name	attribute value
processing directive	processing directive	target of the directive	the content of the directive (w/o target)

Node Types (II)

Node Type	XML component	nodeName	nodeValue
comment	comment	*#comment*	the whole comment
CDATA section	CDATA section	*#cdata-section*	the content of the CDATA section
document type	the document type declaration	root element as used in DOCTYPE declaration	*#null*
entity	an entity (declared within the DTD)	entity name	*#null*
notation	a notation (declared within the DTD)	notation name	*#null*

Using The DOM

- all nodes contain properties and methods that can be used for navigating through and working with the corresponding XML document
  - if a property does not exist for a certain node, it returns *null*
- to access an XML file using the Document Object Model the file must be bound to an (X)HTML document (shown once again for the fictional "library.xml"):
  - `<xml id="dsolibrary" src="library.xml"></xml>`
- the actual implementation depends on the used technology (e.g. script languages like JavaScript, JScript, PHP or PERL)
  - an example using JScript can be found in files_05.zip; it only works with Internet Explorer 6.0 or higher (ActiveX is needed)

Resources

- Carl, D. (2006): Praxiswissen Ajax. O'Reilly.
- Krause, J. (2005): PHP 5 – Grundlagen und Profiwissen. Hanser.
- Shea, D. / Keith, J. (2005): DOM Scripting. APress.
- DOM level 1 specification:
  - http://www.w3.org/tr/rec-dom-level-1
- a list of node objects, properties and methods including the implementation status in common browsers can be found here:
  - http://www.w3schools.com/dom/dom_node.asp

XSLT

- XSLT (Extendable Stylesheet Language Transformations) is a standardised, declarative transformation language to describe and control the transformation of XML documents into other formats, such as HTML, XHTML or text via an XSLT processor
- the XSLT processor converts a source XML document into a result document using an XSL stylesheet document (a sort of template) -> the XSL document is written in XML too

Transformation



XSLT Processors

- MS IE 6.0 or higher
  - includes one
- Oxygen XML editor and debugger for Eclipse
  - http://www.oxygenxml.com/
- Apache Cocoon (including Xalan XSLT processor)
  - http://cocoon.apache.org/
- PHP / Sablotron XSLT support
  - http://at.php.net/xslt
- Saxon XSLT / XQuery processor
  - http://saxon.sourceforge.net/

Resources

- XSLT reference
  - http://www.w3schools.com/xsl/xsl_w3celementref.asp
- specifications
  - http://www.w3.org/Style/XSL/
  - http://www.w3.org/TR/xslt20/
- literature
  - Bongers, F. (2004): XSLT 2.0 - Das umfassende Handbuch zu XSLT 2.0, XPath 2.0 und Saxon 7. Galileo Press.
  - Skulschus, M. / Wiederstein, M. (2005): XSLT und XPath für HTML, Text und XML. Mitp.

Flexibility

- the content of the XML document can be re-structured
- elements, attributes, processing directives, namespaces and comments can be accessed
- the data can be filtered or sorted
- variables and loops can be used
- CSS formatting is available when transforming to HTML / XHTML

- because of many functions XSL is a very complex language

Integrating XML and XSL

- remember the processing directive used for integrating CSS files into XML documents:
  - `<?xml-stylesheet type="text/css" href="`*`filename`*`"?>`
- for XSL files a similar processing directive is set within the XML file:
  - `<?xml-stylesheet type="text/xsl" href="`*`filename`*`"?>`
  - when declaring several XSL stylesheets only the first one is used
  - when declaring XSL and CSS stylesheets only the XSL stylesheet is used
- the components of an XML document are then represented in XSLT by a tree structure (similar to the DOM's node structure); the whole XML document has its equivalent in the XSLT root node

XPath

- for navigation within the tree and accessing the included informations XPath (XML Path Language) is used
- XPath is a path description language for XML documents and a derived subset of XQuery
  - http://www.w3.org/TR/xpath
  - http://www.w3.org/TR/xquery
- XSLT 1.0 only works with XPath 1.0; XSLT 2.0 only works with XPath 2.0
- XPath is mainly used within XSLT

XSL Stylesheet Structure (I)

- file structure
  -
```
<?xml version="1.0"?>
<!- comments -->
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/xsl/transform">

<xsl:template match="/"> (corresponds to the XSLT root node)
 … content …
</xsl:template>

</xsl:stylesheet>
```

XSL Stylesheet Structure (II)

- content structure (transforming to HTML / XHTML)
  - <html>
    <head>
    …
    </head>
    <body>
    <span optional_*CSS_instructions*>*optional_content*
    <xsl:value-of select="*element_name*"/>

    …
    </body>
    </html>

Example – XML Document

- XML document
  -
```
<library>
 <book>
  <author>Mark Twain</author>
  <title>Huckleberry Finn</title>
  <pages>334</pages>
 </book>
 …
</library>
```

Example – XSL Stylesheet

- XSL document
  -
```
<span style="font-style:italic">Author:
 <xsl:value-of select="library/book/author"/>
</span><br />
<span>Title:
 <xsl:value-of select="library/book/title"/>
</span>
```
- the element path must start with the template match node (unless the XSLT processor already moved to another node (the so-called context node) while processing the stylesheet)

Loops (I)

- to show all elements of a data set
  -
```
<xsl:for-each select="library/book">
 <span>Author: <xsl:value-of select="author"/>
 </span><br />
 <span>Title: <xsl:value-of select="title"/>
 </span><br />
 <span>Pages: <xsl:value-of select="pages"/>
 </span>
</xsl:for-each>
```
- now the actual node (context node) is `library/book`, selected in the `xsl:for-each`–statement, therefore only the path from that node onwards has to be specified

Loops (II)

- loops can also be created by defining more than one template in a stylesheet
  -
```
<xsl:template match="/">
 <body>
  <xsl:apply-templates select="library/book"/>
 </body>
</xsl:template>
<xsl:template match="book">
 <span>Author: <xsl:value-of select="author"/>
 </span><br />
 <span>Title: <xsl:value-of select="title"/>
 </span>
</xsl:template>
```

Select- & Match-Terms (I)

Path	Meaning	Example
`name`	the element with the specified name	`book`
`/` (within a path)	separates the levels of a path	`book/title`
`/` (at the beginning of a path)	the XSLT root node	`/library`

Select- & Match-Terms (II)

Path	Meaning	Example
`//`	recursion; the following term means all subordinated elements on any level	`library//author` (means all authors in the library)
`.` (`select` only)	the actual context node	`<xsl:value-of select="."/>` (returns the context node)
`..` (`select` only)	the superordinated node to a context node	`../author` (each `author`-element on the same level as the context node)

Select- & Match-Terms (III)

Path	Meaning	Example
`*`	each element	`book/*` (each element subordinated to `book`)
`@name`	the attribute with the specified name	`book/@available` (each attribute with the specified name belonging to a `book` element)
`|`	combines more than one path in a path	`* | @*` (all element and attribute nodes)

Functions

- a function is an XSLT module that performs a task and then returns a value
  - `<xsl:value-of select="sum(library/books/pages)"/>`
- if at least one of the corresponding nodes returns a value not being a number the function returns "NaN" meaning "not a number"
- for a list of all available functions see:
  - http://www.w3.org/TR/xquery-operators/

Filtering (I)

- a filter defines a condition to narrow down the number of selected nodes
  - `<xsl:for-each select="library/book[author='Mark Twain']">`

    ```
    <span>Title: <xsl:value-of select="title"/>
    </span>
    </xsl:for-each>
    ```
- What about this one?
  - `<xsl:apply-templates select="library/book[author='Mark Twain']"/>`

    ```
    …
    <xsl:template match="book">
     <span>Title: <xsl:value-of select="title"/>
     </span>
    </xsl:template>
    ```

Filtering (II)

- And what about this one?
  - `<xsl:apply-templates select="library/book"/>`

```
...
<xsl:template match="book[author='Mark Twain']">
 <span>Title: <xsl:value-of select="title"/>
 </span>
</xsl:template>
```

Operators To Compare

Operator	Meaning
=	is
!=	is not
<	smaller than
<=	smaller than or same as
>	bigger than
>=	bigger than or same as

- remember: < and <= must be used as < is not a valid character within an attribute's value (see p.17)

Special Filterings

- if there is more than one subordinated element with the same name
  - `<xsl:for-each select="catalog/trousers[colour[2]='blue']">`
- if all subordinated elements of a specific element are to be selected
  - `<xsl:for-each select="library/book[5]/*">`
- a set of subelements may only be selected if it includes a certain subelement
  - `<xsl:for-each select="library/book[pages]">`

Sorting

- controls the order of the nodes
  -
```
<xsl:for-each select="library/book">
 <xsl:sort select="author" data-type="text" order="ascending"/>
 <xsl:sort select="title" data-type="text" order="ascending"/>
 <span>Author (alphabetically): <xsl:value-of select="author"/>
 </span>
 <span>Title: <xsl:value-of select="title"/>
 </span>
</xsl:for-each>
```
- values for `data-type`: `text` and `number`
- values for `order`: `ascending` and `descending`

Accessing Attributes

- displaying a specific attribute of an element
  - `<xsl:value-of select="element/@attribute_name"/>`
- displaying all attributes of an element
  - `<xsl:value-of select="element/@*"/>`
- filtering using an attribute (without its value)
  - `<xsl:for-each select="element[@attribute_name]">`
- filtering using an attribute (with its value)
  - `<xsl:for-each select="element[@attribute_name='value']">`

Conditions (I)

- if-condition
  -
```
<xsl:for-each select="library/book">
 <span>
  <xsl:value-of select="title"/>
  <xsl:if test="@available='no'">Not available!</xsl:if>
 </span>
 <br />
</xsl:for-each>
```

Conditions (II)

- choose-condition
  -
```
<xsl:for-each select="library/book">
 <span>
  <xsl:choose>
   <xsl:when test="pages &lt;=300">*</xsl:when>
   <xsl:when test="pages &lt;=500">**</xsl:when>
   <xsl:otherwise>***</xsl:otherwise>
  </xsl:choose>
  <xsl:value-of select="title"/>
 </span>
 <br />
</xsl:for-each>
```

Chapter II

Semantic Web

The Semantic Web

- is a concept by Tim Berners-Lee to extend the world wibe web with machine-readable data describing the semantics of the content
- the idea is to extend human-readable data with additional information (meta data) that can be interpreted by a machine, so inquiries can be processed according to their meaning, rather than their spelling
- to create these additional data and to implement a semantic web, several methods can be used
  - taxonomies
  - thesauri
  - resource description frameworks
  - ontologies
  - topic maps

Semantic Web – Examples (I)

- FOAF – Friend of a Friend
  - is a project to model a machine-readable social network
  - a FOAF document contains information about a person (such as name, age, gender, addresses (website, weblog, home…), messenger-id…) and other people this person knows
  - the documents then refer to each other and can be analysed by a software, which is then able to visualise the details and the social structure
  - FOAF uses RDF
  - the project: http://www.foaf-project.org/
  - the specification: http://xmlns.com/foaf/0.1/

Semantic Web – Examples (II)

- DOAC – Description of a Career
  - is a project to describe the curriculum of a person in machine-readable form
  - a DOAC document contains information about the skills of a person (such as education, spoken languages, experiences, previous jobs, drivers license
  - it is compatible with the European Unino Europass Curriculum (see http://europass.cedefop.europa.eu/)
  - DOAC uses RDF
  - project: http://ramonantonio.net/doac/
  - specification: http://ramonantonio.net/doac/0.1/

Semantic Web – Examples (III)

- SemanticGov
  - is a project of an international consortium, sponsored by the European Union, to improve the administration of the EU (e.g. the co-operation between authorities)
  - the project duration is 36 months (01.01.2006 – 31.12. 2008), the budget is around 4.37 billion euros
  - project: http://www.semantic-gov.org/

Meta Data

- data containing information about other data (the latter mostly a larger amount of data)
- are used to describe information resources
  - to improve their discovering
  - to document their mutual relations
- are saved
  - within the document (as a meta tag)
  - in assigned reference books (such as catalogs)
  - as an attribute which is held together with the document

Classification

- is a method to divide objects into categories or classes
- the division bases on the moulding of the objects common properties
- can be done
  - manually (categorising, sorting, indexing)
  - automatically (supervised learning)
- classification systems (divided by structure)

	mono-hierarchical	**poly-hierarchical**
heredity	single (strong hierarchy)	multiple (weak hierarchy)
superclasses	one	more than one
structure	tree	non-cyclic directed graph

Taxonomies

- are classification systems with mono-hierarchical structures
- to create a simple semantic
- the root contains general information
- when navigating through the tree structure from the root element on the information gets more and more specific

Thesauri

- are classification systemes with poly-hierarchical structures
- are systematically ordered, networked collections of terms (a so-called "controlled vocabulary", that means a list of terms with an unambiguous, non-redundant definition that have been enumerated explicitly), connected via associative and parent-child relationships
- used to describe/represent topics, for subject indexing and/or document retrieval
- examples:
  - OpenThesaurus: http://www.openthesaurus.de/
  - UNESCO thesaurus: http://databases.unesco.org/thesaurus/

RDF (I)

- the Resource Description Framework (RDF) was developed as a foundation stone for the semantic web by the W3C in 1999
- it is a formal language for the provision of meta data via the WWW and defines a fundamental vocabulary to formulate arbitrary statements about resources
- a statement is a triple consisting of (in this order)
  - a subject
  - a predicate (property)
  - an object
  - -> a person (*subject*) has (*predicate*) a name (*object*)
  - all elements of a triple (the *resources*) are identified by a URI
  - an RDF document is a collection of linked statements

RDF (II)

- the predicate tells something about the subject
- the object is the value of the predicate; it can be a resource (identified by a URI) or just a literal (a constant value)
- predicates and objects can be subjects in different statements
- resources can be used for grouping, they don't have a name then (i.e. a name can be grouped in name and first name)
- statements can be subjects in another statement -> this is called a reification

RDF Graph

- is the standard development method for RDF
- is a labeled, directed, easy readable graph
- modeling:
  - subjects are modeled as a node in form of an ellipse
  - predicates are modeled as an arc
  - objects represented by a URI are modeled as a node in form of an ellipse
  - objects represented by a literal are modeled as a node in form of a box
  - resources used for grouping are modeled as blank nodes (these cannot be referenced)
- SPARQL (SPARQL Protocol and RDF Query Language) is a W3C query language recommendation for RDF graphs
  - http://www.w3.org/TR/rdf-sparql-query/

RDF Graph - Example

- there is a person identified by http://www.w3.org/people/em/contact#me, whose name is Eric Miller, whose email address is em@w3.org, and whose title is Dr.

Blank Nodes – Example (I)



Blank Nodes – Example (II)



Blank Nodes – Example (III)

```
exstaff:85740        exterms:address        exaddressid:85740 .
exaddressid:85740    exterms:street         "1501 Grant Avenue" .
exaddressid:85740    exterms:city           "Bedford" .
exaddressid:85740    exterms:state          "Massachusetts" .
exaddressid:85740    exterms:postalCode     "01730" .
```

Typed Literals




```
exstaff:85740        exterms:age        "27"^^xsd:integer
```

RDF Syntax

- to implement RDF models two different syntaxes exist
  - N3 (Notation 3) by Tim Berners-Lee
    - http://www.w3.org/DesignIssues/Notation3.html
  - an XML based syntax which is the most used
    - http://www.w3.org/TR/rdf-syntax-grammar/

RDF Serialisation

- an RDF graph is encoded as XML elements, attributes, element content and attribute values

- URI references of predicates are written as a combination of a prefix denoting a namespace URI and a local element name (so-called XML QNames)
- URI references of subjects and objects are written as XML attribute values
- literal nodes (which are always object nodes) become XML element text content or attribute values

RDF Serialisation - Example

- there is a person identified by http://www.w3.org/people/em/contact#me, whose name is Eric Miller, whose email address is em@w3.org, and whose title is Dr.
  -
```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:contact="http://www.w3.org/2000/10/swap/pim/contact#">
  <contact:Person rdf:about="http://www.w3.org/People/EM/contact#me">
    <contact:fullName>Eric Miller</contact:fullName>
    <contact:mailbox rdf:resource="mailto:em@w3.org"/>
    <contact:personalTitle>Dr.</contact:personalTitle>
  </contact:Person>
</rdf:RDF>
```

Multiple Statements

- an RDF graph consisting of multiple statements can be represented using multiple description elements:
  -
```
<?xml version="1.0"?>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:dc="http://purl.org/dc/elements/1.1/"
           xmlns:exterms="http://www.example.org/terms/">
    <rdf:Description rdf:about="http://www.example.org/index.html">
      <exterms:creation-date>August 16, 1999</exterms:creation-date>
    </rdf:Description>
    <rdf:Description rdf:about="http://www.example.org/index.html">
      <dc:language>en</dc:language>
    </rdf:Description>
  </rdf:RDF>
```

Multiple Predicates

- a description element may also contain multiple predicates (predicates may even import more than one namespace):
  -
```
<?xml version="1.0"?>
 <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
          xmlns:dc="http://purl.org/dc/elements/1.1/"
          xmlns:exterms="http://www.example.org/terms/">
   <rdf:Description rdf:about="http://www.example.org/index.html">
     <exterms:creation-date>August 16, 1999</exterms:creation-date>
     <dc:language>en</dc:language>
     <dc:creator
rdf:resource="http://www.example.org/staffid/85740"/>
   </rdf:Description>
 </rdf:RDF>
```

Serialisation - Blank Nodes

- blank nodes are serialised using node identifiers
  -
```
<?xml version="1.0"?>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:dc="http://purl.org/dc/elements/1.1/"
           xmlns:exterms="http://example.org/stuff/1.0/">
    <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar">
      <dc:title>RDF/XML Syntax Specification (Revised)</dc:title>
      <exterms:editor rdf:nodeID="abc"/>
    </rdf:Description>
    <rdf:Description rdf:nodeID="abc">
      <exterms:fullName>Dave Beckett</exterms:fullName>
      <exterms:homePage rdf:resource="http://purl.org/net/dajobe/"/>
    </rdf:Description>
  </rdf:RDF>
```

Serialisation: Typed Literals (I)



Serialisation: Typed Literals (II)

- <?xml version="1.0"?>
  - <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:exterms="http://www.example.org/terms/">
    <rdf:Description rdf:about="http://www.example.org/index.html">
    <exterms:creation-date
    rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
    1999-08-16</exterms:creation-date>
    </rdf:Description>
    </rdf:RDF>

Using XML Entities

-
```
<?xml version="1.0"?>
 <!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
   <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:exterms="http://www.example.org/terms/">
     <rdf:Description rdf:about="http://www.example.org/index.html">
       <exterms:creation-date rdf:datatype="&xsd;date">1999-08-16
       </exterms:creation-date>
     </rdf:Description>
   </rdf:RDF>
```

RDF-Containers

- `rdf:Bag`
  - group of resources or literals, possibly including duplicate members, without order
- `rdf:Seq`
  - group of resources or literals, possibly including duplicate members, where the order of the members is significant
- `rdf:Alt`
  - group of resources or literals that are alternatives

RDF-Containers - Example

```
<?xml version="1.0"?>
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:s="http://example.org/students/vocab#">
    <rdf:Description rdf:about="http://example.org/courses/6.001">
      <s:students>
        <rdf:Bag>
          <rdf:li rdf:resource="http://example.org/students/Amy"/>
          <rdf:li rdf:resource="http://example.org/students/Tom"/>
          <rdf:li rdf:resource="http://example.org/students/Jim"/>
        </rdf:Bag>
      </s:students>
    </rdf:Description>
</rdf:RDF>
```

Resources

- literature
  - Powers, S. (2003): Practical RDF. O'Reilly.
- specification
  - http://www.w3.org/RDF/
- validator
  - http://www.w3.org/RDF/Validator/
- RDF/XML syntax specification
  - http://www.w3.org/TR/rdf-syntax-grammar/

RDFS (I)

- the Resource Description Framework Schema (RDFS) is a W3C recommendation used to describe types and properties of resources
- it provides a type system similar to those used in object-oriented programming languages
  - a class hierarchy
  - resources as instances of one or more classes
- RDFS facilities are themselves provided in form of an RDF vocabulary defined in a namespace which is bound to an URI
  - http://www.w3.org/2000/01/rdf-schema#
- vocabulary descriptions written in RDFS always represent valid RDF graphs

RDFS (II)

- a class in RDFS corresponds to the generic concept of a type or category and can represent almost any category of thing, such as web pages, people, document types, databases or abstract concepts
- describing classes:
  - resources: `rdfs:Class`, `rdf:Resource`
  - attributes `rdf:type`, `rdfs:subClassOf`
- describing properties:
  - class: `rdf:Property`
  - properties: `rdfs:domain`, `rdfs:range`, `rdfs:subPropertyOf`

Describing Classes

- "full" description
  - <rdf:Description rdf:ID="class_name">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    </rdf:Description>
- "abbreviated" description
  - <rdfs:Class rdf:ID="class_name"/>
- specialisation of classes
  - <rdfs:Class rdf:ID="class_name">
    <rdfs:subClassOf rdf:resource="super_class"/>
    </rdfs:Class>

Describing Attributes

- properties are described as instances of the class `rdf:Property`
- `rdfs:range` – values of a property are instances of a designated class
- `rdfs:domain` – properties apply to a designated class
- `rdfs:subPropertyOf` – properties can be specialised
- properties may have more than one `rdf:range`, `rdf:domain` and `rdf:subPropertyOf` properties

```
– <rdf:Property rdf:ID="property_name">
    <rdfs:domain rdf:resource="#designated_class"/>
    <rdfs:range rdf:resource="#designated_class"/>
    <rdfs:subPropertyOf rdf:resource="#super_property"/>
  </rdf:Property>
```

RDFS – Example (I)



RDFS – Example (IIa)

- RDF Schema document:
  -
```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd
  "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xml:base="http://example.org/schemas/vehicles">
<rdfs:Class rdf:ID="MotorVehicle"/>
<rdfs:Class rdf:ID="PassengerVehicle">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>
```

RDFS – Example (IIb)

```
<rdfs:Class rdf:ID="Truck">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Van">
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdfs:Class>
<rdfs:Class rdf:ID="MiniVan">
  <rdfs:subClassOf rdf:resource="#Van"/>
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdfs:Class>
<rdfs:Class rdf:ID="Person"/>
<rdfs:Datatype rdf:about="&xsd;integer"/>
```

RDFS – Example (IIc)

```
<rdf:Property rdf:ID="registeredTo">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Property>
<rdf:Property rdf:ID="rearSeatLegRoom">
  <rdfs:domain rdf:resource="#PassengerVehicle"/>
  <rdfs:range rdf:resource="&xsd;integer"/>
</rdf:Property>
<rdf:Property rdf:ID="driver">
  <rdfs:domain rdf:resource="#MotorVehicle"/>
</rdf:Property>
```

RDFS – Example (IId)

```
<rdf:Property rdf:ID="primaryDriver">
  <rdfs:subPropertyOf rdf:resource="#driver"/>
</rdf:Property>
</rdf:RDF>
```

RDFS – Example (III)

- corresponding RDF instance document
  -
```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">]>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex="http://example.org/schemas/vehicles#"
         xml:base="http://example.org/things">
 <ex:PassengerVehicle rdf:ID="johnSmithsCar">
  <ex:registeredTo rdf:resource="http://www.example.org/staffid/85740"/>
  <ex:rearSeatLegRoom rdf:datatype="&xsd;integer">127</ex:rearSeatLegRoom>
  <ex:primaryDriver rdf:resource="http://www.example.org/staffid/85740"/>
 </ex:PassengerVehicle>
</rdf:RDF>
```

Built-In Properties

- RDFS supplies a number of built-in properties
  - `rdfs:comment` - to provide a human-readable description of a resource
  - `rdfs:label` - to provide a more human-readable version of a resource's name
  - `rdfs:seeAlso` - to indicate a resource that might provide additional information about the subject resource
  - `rdfs:isDefinedBy` - to indicate a resource that defines the subject resource (subproperty of `rdfs:seeAlso`)

Dublin Core

- minimal set of descriptive elements that facilitate the description and the automated indexing of document-like networked objects
- originally developed by the Dublin Core Metadata Initiative (DCMI) in March 1995 at a workshop on metadata management in Dublin, Ohio
- i.e. used for the OASIS Open Document Format for Office Applications (OpenDocument) and for RSS 1.0
- imports the namespaces `dc` (http://purl.org/dc/elements/1.1/) and `dcterms` (http://purl.org/dc/terms/)
  - the latter defines additional vocabulary
- of course Dublin Core properties can be declared within an RDFS document

Dublin Core – Elements (I)

Property	Description
title	a name given to the resource
creator	an entity primarily responsible for making the content of the resource
subject	the topic of the content of the resource
description	an account of the content of the resource
publisher	an entity responsible for making the resource available

Dublin Core – Elements (II)

Property	Description
contributor	an entity responsible for making contributions to the content of the resource
date	a date associated with an event in the life cycle of the resource
type	the nature or genre of the content of the resource
format	the physical or digital manifestation of the resource
identifier	an unambiguous reference to the resource within a given context

Dublin Core – Elements (III)

Property	Description
source	a reference to a resource from which the present resource is derived
language	a language of the intellectual content of the resource
relation	a reference to a related resource
coverage	the extent or scope of the content of the resource
rights	information about rights held in and over the resource

Dublin Core - Example

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:dc="http://purl.org/dc/elements/1.1/">
 <rdf:Description rdf:about="http://www.dlib.org">
  <dc:title>D-Lib Program - Research in Digital Libraries</dc:title>
  <dc:description>The D-Lib program supports the community of people with
   research interests in digital libraries and publishing.</dc:description>
  <dc:publisher>Corporation For National Research Initiatives</dc:publisher>
  <dc:date>1995-01-07</dc:date>
  <dc:subject>Research, statistical methods</dc:subject>
  <dc:type>World Wide Web Home Page</dc:type>
  <dc:format>text/html</dc:format>
  <dc:language>en</dc:language>
 </rdf:Description>
</rdf:RDF>
```

Resources

- Dublin Core Metadata Initiative (DCMI)
  - http://dublincore.org/
- expressing Dublin Core in HTML/XHTML meta and link elements
  - http://dublincore.org/documents/dcq-html/
- expressing qualified Dublin Core in RDF / XML
  - http://dublincore.org/documents/dcq-rdf-xml/
- expressing Dublin Core using SelfHTML
  - http://de.selfhtml.org/html/kopfdaten/meta.htm#dublin_core
- RDF test cases
  - http://www.w3.org/TR/rdf-testcases/

Excursion – XBRL (I)

- XBRL (eXtensible Business Reporting Language) is an emerging XML-based standard to define and exchange business and financial performance information. The standard is governed by a non-profit international consortium (XBRL International Incorporated) of approximately 450 organisations, including regulators, government agencies, infomediaries and software vendors.
- XBRL International is supported by its jurisdictions - independent bodies, generally organised on a country-specific basis, that work to promote the adoption of XBRL and the development of taxonomies that define the information exchange requirements of their particular domains. XBRL is being rapidly adopted to replace both paper-based and legacy electronic financial data collection by a wide range of regulators. It is now starting to be used for the disclosure of financial performance information by companies, notably in voluntary (e.g., "trial basis") SEC filings.

Excursion – XBRL (IIa)

- XBRL is a standards-based way to communicate business and financial performance data. These communications are defined by metadata set out in taxonomies. Taxonomies capture the definition of individual reporting elements as well as the relationships between elements within a taxonomy and in other taxonomies. The U.S. Federal Deposit Insurance Corporation (FDIC), in coordination with the Federal Reserve Board and the Office of the Controller of the Currency, launched a large and very successful XBRL project in October 2005 involving the collection of quarterly bank financial statements (Call Reports) from over 8300 U.S. banks. Use of XBRL is mandatory and the data is posted on the Internet for public use and analysis. This coordinated U.S. project has proven that XBRL can provide real business value by reducing burden and duplication, improving data transparency and enabling more timely analysis. However, the FDIC implementation of XBRL is a limited example in that the banks use a form-based template to submit their data, a template that actually predates XBRL by almost two decades.

Excursion – XBRL (IIb)

- The FDIC implementation of XBRL remains the largest use of XBRL in the U.S.. A white paper was prepared in early 2006 describing the project and business results.
- Historical US company SEC filings information can be downloaded from Edgar Online on a subscription basis. News distribution companies such as PR Newswire and Business Wire provide services to listed companies to allow them to distribute their financial information in XBRL format for a fee.
- The first European project for the Dutch Waterboards, and further successful projects in Europe including Spain, Belgium and others, have given Europe a leading role in implementations. Currently a major project of the Dutch Government for XBRL reporting by all businesses as well as (semi-)government organisations like cities and health institutes will make it the first nation-wide implementation.

XBRL – Document Structure

- XBRL consists of an instance document, containing primarily the business facts being reported, and a collection of taxonomies, which define metadata about these facts, such as what the facts mean and how they relate to one another. XBRL uses XML schema, XLink, and XPointer standards.

XBRL – Instance Document

- The instance document holds the <xbrl> root. The document itself holds the following information:
  - *Business Facts* - facts can be divided into two categories
    - *Items* are facts holding a single value. They are represented by a single XML element with the value as its content.
    - *Tuples* are facts holding multiple values. They are represented by a single XML element containing nested Items.
  - *Contexts* define the entity (i.e. company or individual) to which the fact applies, the period of time the fact is relevant, and an optional scenario. Scenarios provide further contextual information about the facts, such as whether the business values reported are actual, projected, budgeted, etc.
  - *Units* define the units used by numeric or fractional facts within the document, such as USD, shares. XBRL allows more complex units to be defined if necessary.
  - *Footnotes*
  - *References* to taxonomies, typically through schema references

XBRL - Taxonomies

- Taxonomies are a collection of XML schema documents and XML documents called linkbases by virtue of their use of XLink. The schema must ultimately extend the XBRL instance schema document and typically extend other published XBRL schemas on the xbrl.org website.
- *Schemas* define Item and Tuple "concepts" using <xsd:element> elements. Concepts provide names for the fact and indicate whether or not it's a tuple or item, what type of data it contains (monetary, numeric, fractional, textual, etc.) among some other metadata. Items and Tuples can be regarded as "implementations" of concepts, or specific instances of a concept. In addition to defining concepts, Schemas reference linkbase documents.
- *Linkbases* are a collection of links, which themselves are a collection of locators, arcs, and potentially resources. Locators are elements that essentially reference a concept and provide an arbitrary label for it. In turn, arcs are elements indiciating that a concept links to another concept by referencing the lables defined by the locators. Some arcs link concepts to other concepts. Other arcs link concepts to resources, the most common of which are human-readable labels for the concepts.

Resources

- Watson, L.; Vasarhelyi, M. (2003): Essentials of XBRL: Financial Reporting in the 21st Century. Wiley & Sons.
- the official XBRL website
  - http://www.xbrl.org/home/
- the XBRL specification
  - http://www.xbrl.org/SpecRecommendations/
- worldwide XBRL projects
  - http://serverlab.unab.edu.co:8080/mediawiki/index.php/XBRL_Projects
- FDIC white paper on business value (PDF)
  - http://www.xbrl.org/us/us/FFIEC%20White%20Paper%2002Feb2006.pdf
- ABRA – open source program to process XBRL
  - http://www.xbrlopen.org/abra/index.html

Ontologies

- are data models that represent a domain and are used to reason about the objects in that domain and the relations between them
  - this applies only to computer and information science
- ontologies generally describe:
  - individuals: the basic or "ground level" objects
  - classes: sets, collections, or types of objects
  - attributes: properties, features, characteristics, or parameters that objects can have and share
  - relations: ways that objects can be related to one another
- to encode ontologies, formal ontology languages are used

Individuals

- Individuals (instances) are the basic, "ground level" components of an ontology. The individuals in an ontology may include concrete objects such as people, animals, tables, automobiles, molecules, and planets, as well as abstract individuals such as numbers and words. Strictly speaking, an ontology need not include any individuals, but one of the general purposes of an ontology is to provide a means of classifying individuals, even if those individuals are not explicitly part of the ontology.

Classes

- Classes are abstract groups, sets, or collections of objects. They may contain individuals, other classes, or a combination of both. Some examples of classes:
  - **person**, the class of all people
  - **number**, the class of all numbers
  - **vehicle**, the class of all vehicles
  - **car**, the class of all cars
  - **individual**, representing the class of all individuals
  - **class**, representing the class of all classes
  - **thing**, representing the class of all things

Attributes

- Objects in the ontology can be described by assigning attributes to them. Each attribute has at least a name and a value, and is used to store information that is specific to the object it is attached to. For example the Ford Explorer object has attributes such as:
  - *Name*: Ford Explorer
  - *Number-of-doors*: 4
  - *Engine*: {4.0L, 4.6L}
  - *Transmission*: 6-speed
- The value of an attribute can be a complex data type.

Relationships

- An important use of attributes is to describe the *relationships* (or simply *relations*) between objects in the ontology. Typically a relation is an attribute whose value is another object in the ontology. For example in the ontology that contains the Ford Explorer and the Ford Bronco, the Ford Bronco object might have the following attribute:
  - *Successor*: Ford Explorer
- This means that the Explorer is the model that replaced the Bronco. Much of the power of ontologies comes from the ability to describe these relations. Together, the set of relations describes the semantics of the domain.

Domain vs. Upper Ontologies

- domain ontologies (or doamin-specific ontologies)
  - model a specific domain, or part of the world
  - represent the particular meanings of terms as they apply to that domain
- upper ontologies (or foundation ontologies)
  - are models of the common objects that are generally applicable across a wide range of domain ontologies
  - contain a core glossary in whose terms objects in a set of domains can be described, e.g Dublin Core

Partitions (I)

- A partition is a set of related classes and associated rules that allow objects to be placed into the appropriate class. For example, the partial diagram of an ontology has a partition of the Car class into the classes 2-Wheel Drive and 4-Wheel Drive.
- The partition rule determines if a particular car is placed in the 2-Wheel Drive or the 4-Wheel Drive class.



Partitions (II)

- If the partition rule(s) guarantee that a single Car object cannot be in both classes, then the partition is called a *Disjoint Partition*. If the partition rules ensure that every concrete object in the super-class is an instance of at least one of the partition classes, then the partition is called an *Exhaustive Partition*.

Ontology – Example (I)



Ontology – Example (II)



Web Ontology Language (OWL)

- OWL is part of the growing stack of W3C recommendations related to the Semantic Web:
  - XML provides a surface syntax for structured documents, but imposes no semantic constraints on the meaning of these documents
  - XML Schema is a language for restricting the structure of XML documents and also extends XML with datatypes
  - RDF is a datamodel for objects (i.e. resources) and relations between them, provides a simple semantic for this datamodel, and these datamodels can be represented in an XML syntax
  - RDFS is a vocabulary for describing properties and classes of RDF resources, with a semantics for generalization-hierarchies of such properties and classes
  - OWL adds more vocabulary for describing properties and classes (such as relations between classes, cardinality, equality, richer typing of properties…)

OWL Sublanguages

- OWL provides three increasingly expressive sublanguages:
  - OWL Lite supports classification hierarchies and simple constraints
  - OWL DL allows for maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computable) and decidability (all computations will finish in finite time)
  - OWL Full allows for maximum expressiveness and the syntactic freedom of RDF with no computational guarantees
- each of these sublanguages is an extension of its simpler predecessor:
  - every legal OWL Lite ontology is a legal OWL DL ontology
  - every legal OWL DL ontology is a legal OWL Full ontology
  - every valid OWL Lite conclusion is a valid OWL DL conclusion
  - every valid OWL DL conclusion is a valid OWL Full conclusion

OWL Lite Overview (I)

- RDF Schema Features:
  - Class (Thing, Nothing)
  - rdfs:subClassOf
  - rdf:Property
  - rdfs:subPropertyOf
  - rdfs:domain
  - rdfs:range
  - Individual
- (In)Equality:
  - equivalentClass
  - equivalentProperty
  - sameAs
  - differentFrom
  - AllDifferent
  - distinctMembers
- Property Characteristics:
  - ObjectProperty
  - DatatypeProperty
  - inverseOf
  - TransitiveProperty
  - SymmetricProperty
  - FunctionalProperty
  - InverseFunctionalProperty

OWL Lite Overview (II)

- Property Restrictions:
  - Restriction
  - onProperty
  - allValuesFrom
  - someValuesFrom
- Restricted Cardinality:
  - minCardinality
  - maxCardinality
  - cardinality
- Header Information:
  - Ontology
  - imports
- Class Intersection:
  - intersectionOf
- Datatypes:
  - xsd datatypes

OWL Lite Overview (III)

- Versioning:
  - versionInfo
  - priorVersion
  - backwardCompatibleWith
  - incompatibleWith
  - DeprecatedClass
  - DeprecatedProperty
- Annotation Properties:
  - rdfs:label
  - rdfs:comment
  - rdfs:seeAlso
  - rdfs:isDefinedBy
  - AnnotationProperty
  - OntologyProperty

OWL DL And Full Overview

- Class Axioms:
  - oneOf
  - disjointWith
  - equivalentClass (applied to class expressions)
  - rdfs:subClassOf (applied to class expressions)
- Boolean Combinations of Class Expressions:
  - unionOf
  - complementOf
  - intersectionOf
- Arbitrary Cardinality:
  - minCardinality
  - maxCardinality
  - cardinality
- Filler Information:
  - hasValue

OWL Vocabulary (I)

- Class
- a group of individuals that belong together because they share some properties
  - classes can be organized in a specialisation hierarchy using subClassOf
  - there is a built-in most general class named Thing that is the class of all individuals and is a superclass of all OWL classes
  - there is also a built-in most specific class named Nothing that is the class that has no instances and is a subclass of all OWL classes
- example:
  - ```
    <owl:Class rdf:ID="Winery"/>
    <owl:Class rdf:ID="Region"/>
    <owl:Class rdf:ID="ConsumableThing"/>
    ```
    ```

OWL Vocabulary (II)

- rdfs:subClassOf
  - class hierarchies may be created by making one or more statements that a class is a subclass of another class
  - may only be used in conjunction with single class names (as opposed to OWL DL and Full)
- example:
  -
```
<owl:Class rdf:ID="Wine">
<rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
<rdfs:label xml:lang="fr">vin</rdfs:label>
</owl:Class>
<owl:Class rdf:ID="Pasta">
<rdfs:subClassOf rdf:resource="#EdibleThing"/>
</owl:Class>
```

OWL Vocabulary (III)

- rdf:Property
  - properties can be used to state relationships between individuals (owl:ObjectProperty) or from individuals to XSD data values and RDF literals (owl:DatatypeProperty)
  - owl:ObjectProperty and owl:DatatypeProperty are subclasses of the RDF class rdf:Property
- example:
  -
```
<owl:ObjectProperty rdf:ID="madeFromGrape">
<rdfs:domain rdf:resource="#Wine"/>
<rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
```

OWL Vocabulary (IV)

- rdfs:subPropertyOf
  - property hierarchies may be created by making one or more statements that a property is a subproperty of one or more other properties
- example:
  -
```
<owl:ObjectProperty rdf:ID="hasWineDescriptor">
<rdfs:domain rdf:resource="#Wine" />
<rdfs:range rdf:resource="#WineDescriptor"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasColor">
<rdfs:subPropertyOf rdf:resource="#hasWineDescriptor"/>
<rdfs:range rdf:resource="#WineColor"/>
</owl:ObjectProperty>
```

OWL Vocabulary (V)

- rdfs:domain
  - a domain of a property limits the individuals to which the property can be applied
- example:
  -
```
<owl:ObjectProperty rdf:ID="madeFromGrape">
<rdfs:domain rdf:resource="#Wine"/>
<rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
```

OWL Vocabulary (VI)

- rdfs:range
  - the range of a property limits the individuals that the property may have as its value
- example:
  -
```
<owl:ObjectProperty rdf:ID="madeFromGrape">
<rdfs:domain rdf:resource="#Wine"/>
<rdfs:range rdf:resource="#WineGrape"/>
</owl:ObjectProperty>
```

OWL Vocabulary (VII)

- Individual
  - individuals are instances of classes, and properties may be used to relate one individual to another
- example:
  -
```
<owl:Thing rdf:ID="CentralCoastRegion"/>
<owl:Thing rdf:about="#CentralCoastRegion">
<rdf:type rdf:resource="#Region"/>
</owl:Thing>

<WineGrape rdf:ID="CabernetSauvignonGrape"/>
```

OWL Vocabulary (VIII)

- equivalentClass
  - two classes may be stated to be equivalent
  - may only be used in conjunction with single class names (as opposed to OWL DL and Full)
  - equivalent classes have the same instances
  - equality can be used to create synonymous classes
- example:
  - ```
    <owl:Class rdf:ID="Wine">
    <owl:equivalentClass rdf:resource="&vin;Wine"/>
    </owl:Class>
    ```
    ```

OWL Vocabulary (IX)

- equivalentProperty
  - two properties may be stated to be equivalent
  - equivalent properties have the same range and domain
  - equality can be used to create synonymous properties
- example:
  - ```
    <rdf:Property rdf:ID="weight">
    <owl:equivalentProperty rdf:resource="&ex;weight"/>
    </rdf:Property>
    ```
    ```

OWL Vocabulary (X)

- sameAs
  - two individuals may be stated to be the same
  - these constructs may be used to create a number of different names that refer to the same individual
- example:
  - ```
    <Wine rdf:ID="MikesFavoriteWine">
    <owl:sameAs rdf:resource="#TexasWhite"/>
    </Wine>
    ```
    ```

OWL Vocabulary (XI)

- differentFrom
  - an individual may be stated to be different from other individuals
- example:
  -
```
<WineSugar rdf:ID="Dry"/>

<WineSugar rdf:ID="Sweet">
<owl:differentFrom rdf:resource="#Dry"/>
</WineSugar>
```

OWL Vocabulary (XII)

- AllDifferent
  - a number of individuals may be stated to be mutually distinct in one statement
  - particularly useful when there are sets of distinct objects and when modelers are interested in enforcing the unique names assumption within those sets of objects
- example:
  -
```
<owl:AllDifferent>
<owl:distinctMembers rdf:parseType="Collection">
<vin:WineColor rdf:about="#Red"/>
<vin:WineColor rdf:about="#White"/>
</owl:distinctMembers>
</owl:AllDifferent>
```

OWL Vocabulary (XIII)

- distinctMembers
  - all members of a list are distinct and pairwise disjoint
  - can only be used in combination with owl:AllDifferent
- example:
  -
```
<owl:AllDifferent>
<owl:distinctMembers rdf:parseType="Collection">
<vin:WineColor rdf:about="#Red"/>
<vin:WineColor rdf:about="#White"/>
</owl:distinctMembers>
</owl:AllDifferent>
```

OWL Vocabulary (XIV)

- inverseOf
  - one property may be stated to be the inverse of another property: P1(x, y) => P2(y, x)
- example:
  -
```
<owl:ObjectProperty rdf:ID="hasMaker">
<rdf:type rdf:resource="&owl;FunctionalProperty"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="producesWine">
<owl:inverseOf rdf:resource="#hasMaker"/>
</owl:ObjectProperty>
```

OWL Vocabulary (XV)

- TransitiveProperty
  - properties may be stated to be transitive: P(x, y) ^ P(y, z) => P(x, z)
- example:
  - ```
    <owl:ObjectProperty rdf:ID="locatedIn">
    <rdf:type rdf:resource="&owl;TransitiveProperty"/>
    <rdfs:domain rdf:resource="&owl;Thing"/>
    <rdfs:range rdf:resource="#Region"/>
    </owl:ObjectProperty>
    ```
    ```

OWL Vocabulary (XVI)

- SymmetricProperty
  - properties may be stated to be symmetric: P(x, y) => P(y, x)
- example:
  -
```
<owl:ObjectProperty rdf:ID="adjacentRegion">
<rdf:type rdf:resource="&owl;SymmetricProperty"/>
<rdfs:domain rdf:resource="#Region"/>
<rdfs:range rdf:resource="#Region"/>
</owl:ObjectProperty>
```

OWL Vocabulary (XVII)

- FunctionalProperty
  - if a property is a owl:FunctionalProperty, then it has no more than one value for each individual (unique property): P(x, y) ^ P(x, z) => y = z
  - shorthand for stating that the property's minimum cardinality is zero and its maximum cardinality is 1
- example:
  - ```
    <owl:ObjectProperty rdf:ID="hasVintageYear">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:domain rdf:resource="#Vintage"/>
    <rdfs:range rdf:resource="#VintageYear"/>
    </owl:ObjectProperty>
    ```
    ```

OWL Vocabulary (XVIII)

- InverseFunctionalProperty
  - if a property is inverse functional then the inverse of the property is functional (unambiguous property): P(y, x) ^ P(z, x) => y = z
- example:
  - ```
    <owl:ObjectProperty rdf:ID="producesWine">
    <rdf:type rdf:resource="&owl;InverseFunctionalProperty"/>
    <owl:inverseOf rdf:resource="#hasMaker"/>
    </owl:ObjectProperty>
    ```
    ```

OWL Vocabulary (XIX)

- Restriction
  - for placing restrictions on the usage of properties by class instances
  - applies to its containing class definition only
- example:
  - ```
    <owl:Class rdf:ID="Wine">
    <rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
    <rdfs:subClassOf>
    <owl:Restriction>
    <!-- .... -->
    </owl:Restriction>
    </rdfs:subClassOf>
    </owl:Class>
    ```
    ```

OWL Vocabulary (XX)

- onProperty
  - indicates the restricted property
- example:
  -
```
<owl:Class rdf:ID="Wine">
<rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasMaker"/>
<!-- .... -->
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

OWL Vocabulary (XXI)

- allValuesFrom
  - stated on a property with respect to a class (local range restriction)
  - values of the property are all members of the class indicated
- example:

```
<owl:Class rdf:ID="Wine">
<rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasMaker"/>
<owl:allValuesFrom rdf:resource="#Winery"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

OWL Vocabulary (XXII)

- someValuesFrom
  - stated on a property with respect to a class (local range restriction)
  - at least one value for that property is of a certain type
- example:
  -
```
<owl:Class rdf:ID="Wine">
<rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasMaker"/>
<owl:someValuesFrom rdf:resource="#Winery"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

OWL Vocabulary (XXIII)

- minCardinality
  - stated on a property with respect to a particular class
  - permits the specification of the minimum number of elements in a relation (0 or 1)
- example:
  -
```
<owl:Restriction>
<owl:onProperty rdf:resource="#hasMother"/>
<owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1
</owl:minCardinality>
</owl:Restriction>
```

OWL Vocabulary (XXIV)

- maxCardinality
  - stated on a property with respect to a particular class
  - permits the specification of the maximum number of elements in a relation (0 or 1)
- example:
  -
```
<owl:Restriction>
<owl:onProperty rdf:resource="#hasMother"/>
<owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">1
</owl:maxCardinality>
</owl:Restriction>
```

OWL Vocabulary (XXV)

- cardinality
  - permits the specification of exactly the number of elements in a relation (0 or 1)
- example:
  -
```
<owl:Class rdf:ID="Vintage">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasVintageYear"/>
<owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">1
</owl:cardinality>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

OWL Vocabulary (XXVI)

- Ontology
  - an ontology is a resource and may be described using properties
  - if the value of rdf:about is empty, the name of the ontology is the base URI of the owl:Ontology element
- example:
  -
```
<owl:Ontology rdf:about="">
<owl:versionInfo>v 1.17 2003/02/26 12:56:51 mdean</owl:versionInfo>
<rdfs:comment>Comments are used to annotate ontologies.
</rdfs:comment>
</owl:Ontology>
```

OWL Vocabulary (XXVII)

- imports
  - references another OWL ontology containing definitions, whose meaning is considered to be part of the meaning of the importing ontology
  - owl:imports is a property with the class owl:Ontology as its domain and range
  - imports are transitive
- example:
  - ```
    <owl:Ontology rdf:about="">
    <!-- .... -->
    <owl:imports rdf:resource="http://www.example.org/foo"/>
    </owl:Ontology>
    ```
    ```

OWL Vocabulary (XXVIII)

- intersectionOf
  - allows for intersections of named classes and restrictions
- example:
  -
```
<owl:Class rdf:ID="WhiteWine">
<owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#Wine"/>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasColor"/>
<owl:hasValue rdf:resource="#White"/>
</owl:Restriction>
</owl:intersectionOf>
</owl:Class>
```

OWL Vocabulary (XXIX)

- xsd datatypes
  - OWL uses most of the built-in XML schema datatypes
- example:
  -
```
<owl:Class rdf:ID="VintageYear"/>

<owl:DatatypeProperty rdf:ID="yearValue">
<rdfs:domain rdf:resource="#VintageYear"/>
<rdfs:range rdf:resource="&xsd;positiveInteger"/>
</owl:DatatypeProperty>
```

OWL Vocabulary (XXX)

- versionInfo
  - a string giving information about the version of the corresponding OWL construct
- example:
  -
```
<owl:Ontology rdf:about="">
<owl:versionInfo>v 1.17 2003/02/26 12:56:51 mdean</owl:versionInfo>
</owl:Ontology>
```

OWL Vocabulary (XXXI)

- priorVersion
  - reference to another ontology that identifies the specified ontology as a prior version of the containing ontology
- example:
  -
```
<owl:Ontology rdf:about="">
<owl:priorVersion rdf:resource="http://www.example.org/old"/>
</owl:Ontology>
```

OWL Vocabulary (XXXII)

- backwardCompatibleWith
  - reference to another ontology that identifies the specified ontology as a prior version of the containing ontology, and further indicates that it is backward compatible with it
- example:
  -
```
<owl:Ontology rdf:about="">
<owl:backwardCompatibleWith
rdf:resource="http://www.example.org/old"/>
</owl:Ontology>
```

OWL Vocabulary (XXXIII)

- incompatibleWith
  - reference to another ontology that identifies the specified ontology as a prior version of the containing ontology, and further indicates that is not backward compatible with it
- example:
  -
```
<owl:Ontology rdf:about="">
<owl:incompatibleWith rdf:resource="http://www.example.org/old"/>
</owl:Ontology>
```

OWL Vocabulary (XXXIV)

- DeprecatedClass
  - class is preserved for backward-compatibility purposes, but may be phased out in the future
- example:
  -
```
<owl:DeprecatedClass rdf:ID="Car">
<rdfs:comment>Automobile is now preferred</rdfs:comment>
<owl:equivalentClass rdf:resource="#Automobile"/>
</owl:DeprecatedClass>
```

OWL Vocabulary (XXXV)

- DeprecatedProperty
  - property is preserved for backward-compatibility purposes, but may be phased out in the future
- example:
  -
```
<owl:DeprecatedProperty rdf:ID="hasDriver">
<rdfs:comment>inverse property drives is now preferred
</rdfs:comment>
<owl:inverseOf rdf:resource="#drives" />
</owl:DeprecatedProperty>
```

OWL Vocabulary (XXXVI)

- AnnotationProperty
  - declares a property that is used as an annotation
  - the object of an annotation property must be either a data literal, a URI reference, or an individual
- example:
  - `<owl:AnnotationProperty rdf:about="&dc;creator"/>`

```
<owl:Class rdf:about="#MusicalWork">
<rdfs:label>Musical work</rdfs:label>
<dc:creator>N.N.</dc:creator>
</owl:Class>
```

OWL Vocabulary (XXXVII)

- OntologyProperty
  - properties that apply to the ontology as a whole
  - instances of this class must have the class owl:Ontology as their domain and range
- example:
  -
```
<rdf:Property rdf:ID="incompatibleWith">
<rdfs:label>incompatibleWith</rdfs:label>
<rdf:type rdf:resource="#OntologyProperty"/>
<rdfs:domain rdf:resource="#Ontology"/>
<rdfs:range rdf:resource="#Ontology"/>
</rdf:Property>
```

OWL Vocabulary (XXXVIII)

- oneOf
  - classes can be described by enumeration of the individuals that make up the class, i. e., the members of the class are exactly the set of enumerated individuals
- example:
  -
```
<owl:Class rdf:ID="WineColor">
<rdfs:subClassOf rdf:resource="#WineDescriptor"/>
<owl:oneOf rdf:parseType="Collection">
<owl:Thing rdf:about="#White"/>
<owl:Thing rdf:about="#Rose"/>
<owl:Thing rdf:about="#Red"/>
</owl:oneOf>
</owl:Class>
```

OWL Vocabulary (XIL)

- disjointWith
  - classes may be stated to be disjoint from each other
  - such statements are mainly used to avoid inconsistencies
- example:
  -
```
<owl:Class rdf:ID="Pasta">
<rdfs:subClassOf rdf:resource="#EdibleThing"/>
<owl:disjointWith rdf:resource="#Meat"/>
<owl:disjointWith rdf:resource="#Fowl"/>
<owl:disjointWith rdf:resource="#Seafood"/>
<owl:disjointWith rdf:resource="#Dessert"/>
<owl:disjointWith rdf:resource="#Fruit"/>
</owl:Class>
```

OWL Vocabulary (XL)

- equivalentClass
  - same meaning as in OWL Lite
  - may additionally be complex class definitions
- example:
  -
```
<owl:Class rdf:ID="TexasThings">
<owl:equivalentClass>
<owl:Restriction>
<owl:onProperty rdf:resource="#locatedIn"/>
<owl:someValuesFrom rdf:resource="#TexasRegion"/>
</owl:Restriction>
</owl:equivalentClass>
</owl:Class>
```

OWL Vocabulary (XLI)

- rdfs:subClassOf
  - same meaning as in OWL Lite
  - may additionally be complex class definitions
- example:
  -
```
<owl:Class rdf:ID="Wine">
<rdfs:subClassOf rdf:resource="&food;PotableLiquid"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#madeFromGrape"/>
<owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
1</owl:minCardinality>
</owl:Restriction>
</rdfs:subClassOf>
```

OWL Vocabulary (XLII)

- unionOf
  - boolean combination of classes and/or restrictions
- example:
  -
```
<owl:Class rdf:ID="Fruit">
<owl:unionOf rdf:parseType="Collection">
<owl:Class rdf:about="#SweetFruit"/>
<owl:Class rdf:about="#NonSweetFruit"/>
</owl:unionOf>
</owl:Class>
```

OWL Vocabulary (XLIII)

- complementOf
  - boolean combination of classes and/or restrictions
- example:
  -
```
<owl:Class rdf:ID="ConsumableThing"/>


<owl:Class rdf:ID="NonConsumableThing">
<owl:complementOf rdf:resource="#ConsumableThing"/>
</owl:Class>
```

OWL Vocabulary (XLIV)

- intersectionOf
  - boolean combination of classes and/or restrictions
- example:
  -
```
<owl:Class rdf:ID="WhiteWine">
<owl:intersectionOf rdf:parseType="Collection">
<owl:Class rdf:about="#Wine"/>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasColor"/>
<owl:hasValue rdf:resource="#White"/>
</owl:Restriction>
</owl:intersectionOf>
</owl:Class>
```

OWL Vocabulary (XLV)

- minCardinality
  - stated on a property with respect to a particular class
  - specifies the minimum number of elements in a relation
- example:
  -
```
<owl:Class rdf:ID="Tricycle">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasWheels"/>
<owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
3</owl:minCardinality>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

OWL Vocabulary (XLVI)

- maxCardinality
  - stated on a property with respect to a particular class
  - specifies the maximum number of elements in a relation
- example:
  -
```
<owl:Class rdf:ID="Tricycle">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasWheels"/>
<owl:maxCardinality rdf:datatype="&xsd;nonNegativeInteger">
3</owl:maxCardinality>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

OWL Vocabulary (XLVII)

- cardinality
  - permits the specification of exactly the number of elements in a relation
- example:
  -
```
<owl:Class rdf:ID="Tricycle">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasWheels"/>
<owl:cardinality rdf:datatype="&xsd;nonNegativeInteger">
3</owl:cardinality>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

OWL Vocabulary (XLVIII)

- hasValue
  - a property can be required to have a certain individual as a value
  - allows for the specification of classes based on the existence of particular property values
- example:
  -
```
<owl:Class rdf:ID="Burgundy">
<rdfs:subClassOf>
<owl:Restriction>
<owl:onProperty rdf:resource="#hasSugar"/>
<owl:hasValue rdf:resource="#Dry"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

Resources

- W3C OWL Technical Report
  - http://www.w3.org/TR/owl-features/
- W3C OWL Language Guide
  - http://www.w3.org/TR/owl-guide/
- W3C OWL Language Reference
  - http://www.w3.org/TR/owl-ref/
- BORG – Bremen Ontology Research Group
  - http://www.fb10.uni-bremen.de/ontology/
- Protégé – open source ontology editor with OWL plugin
  - http://protege.stanford.edu/
- Metatomix m3t4.Studio Semantic Toolkit — RDF/OWL editor for Eclipse
  - http://www.m3t4.com

Chapter III

Topic Maps

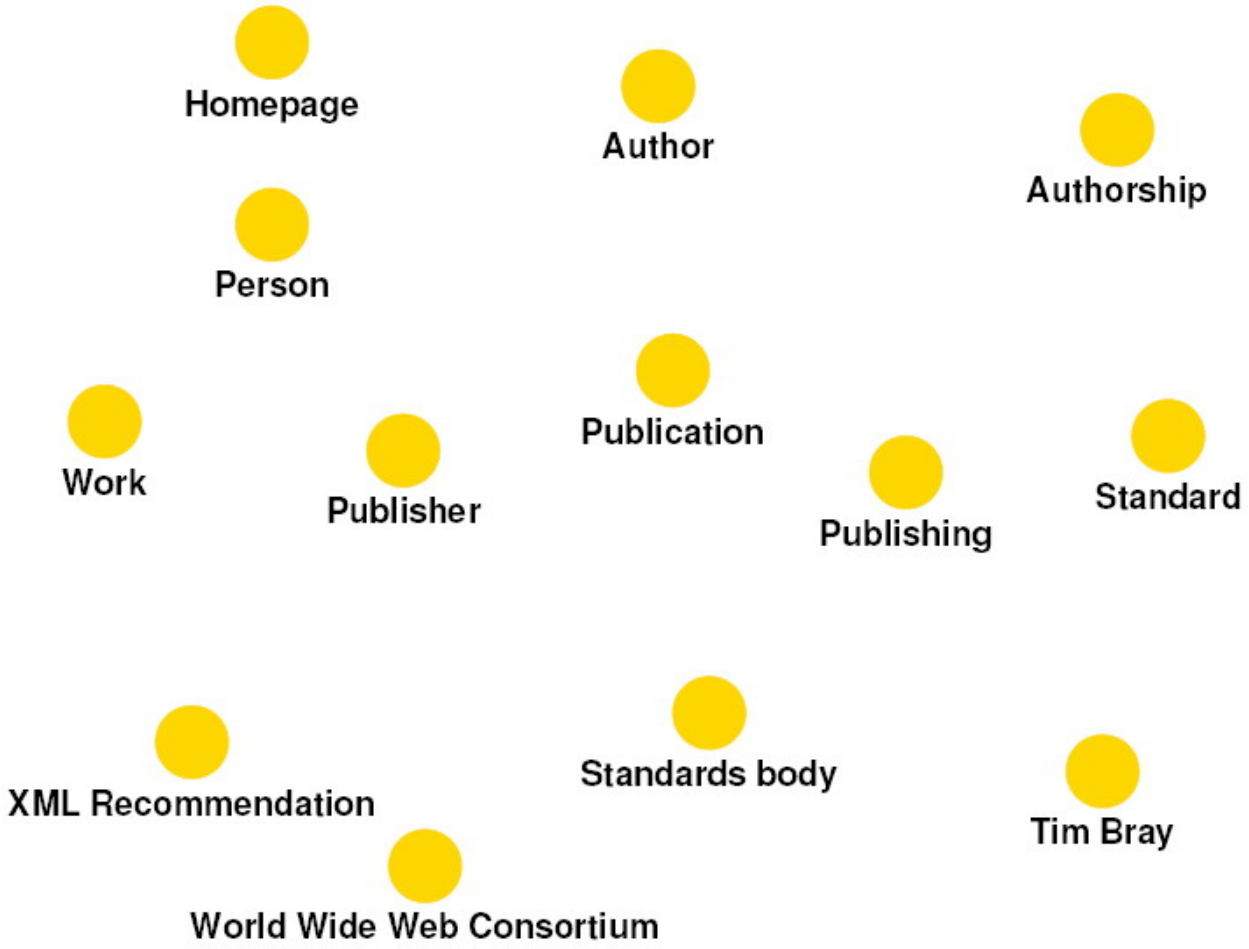
Topic Maps

- ISO standard (ISO/IEC 13250) for an implementation-independent representation of knowledge about resources, their subjects and interrelationships
- topic maps consist of topics (concepts), associations (relationships) and occurrences (relevant information resources)
- as opposed to RDF that aims at machine-processable metadata, topic maps are used to structure knowledge for human readers, with an emphasis on the findability of information
- stem from glossaries, classification systems and thesauri, but provide more expressiveness
- can be used to develop ontologies which may be even mapped to RDF, but are not part of the semantic web effort of the W3C

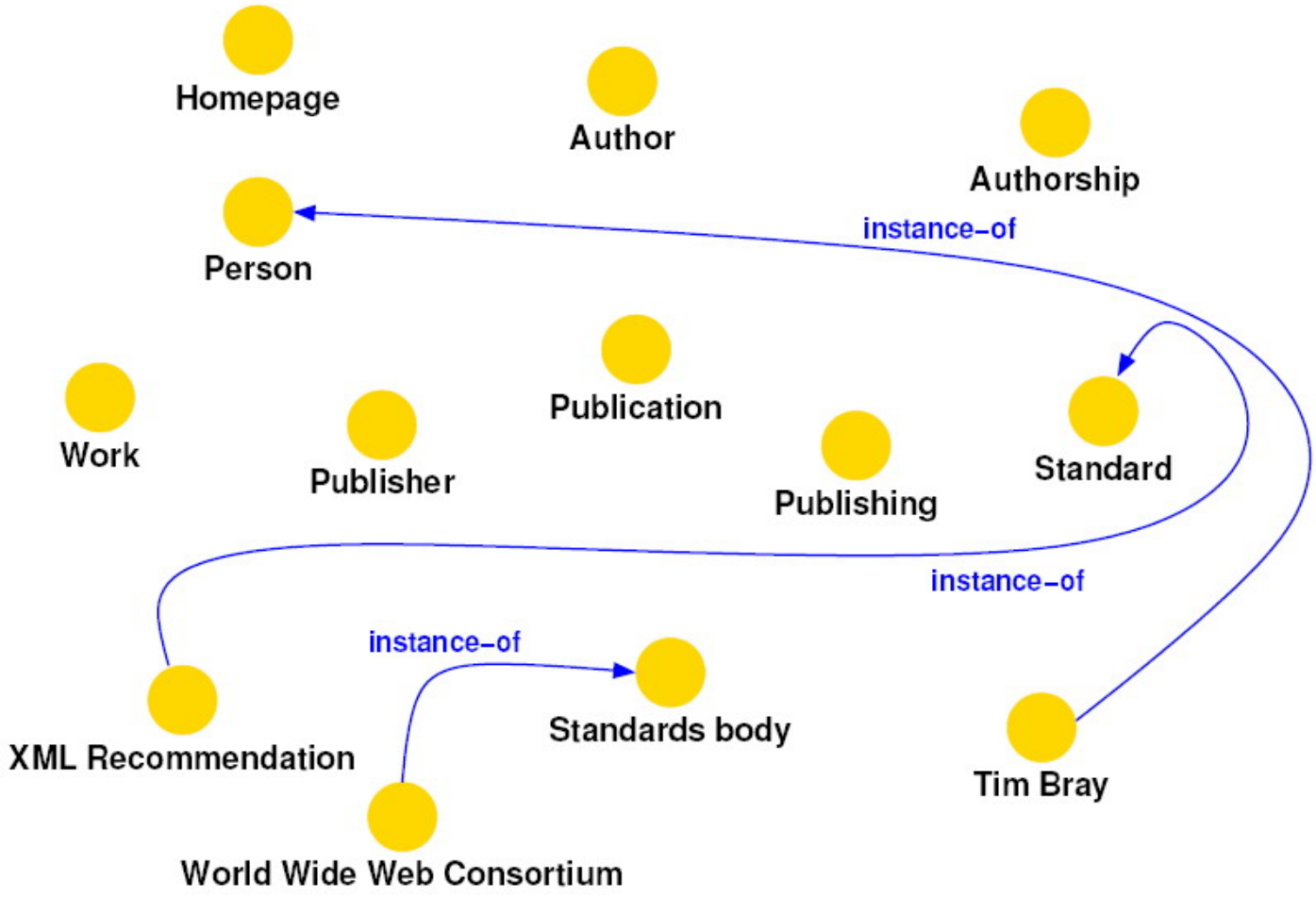
Fundamental Concepts

- topic -> fundamental entity in the context of the modeled knowledge domain
- topic name -> topic identifier (base name, display name and sort name)
- topic occurrence -> instances and roles (occurrence role type)
- public subject descriptor -> unique topic descriptor
- associations -> relationships between topics and their roles (association role)
- scope -> specifies the extent of the validity
- facet -> attribute-value-pair that describes a topic in more detail

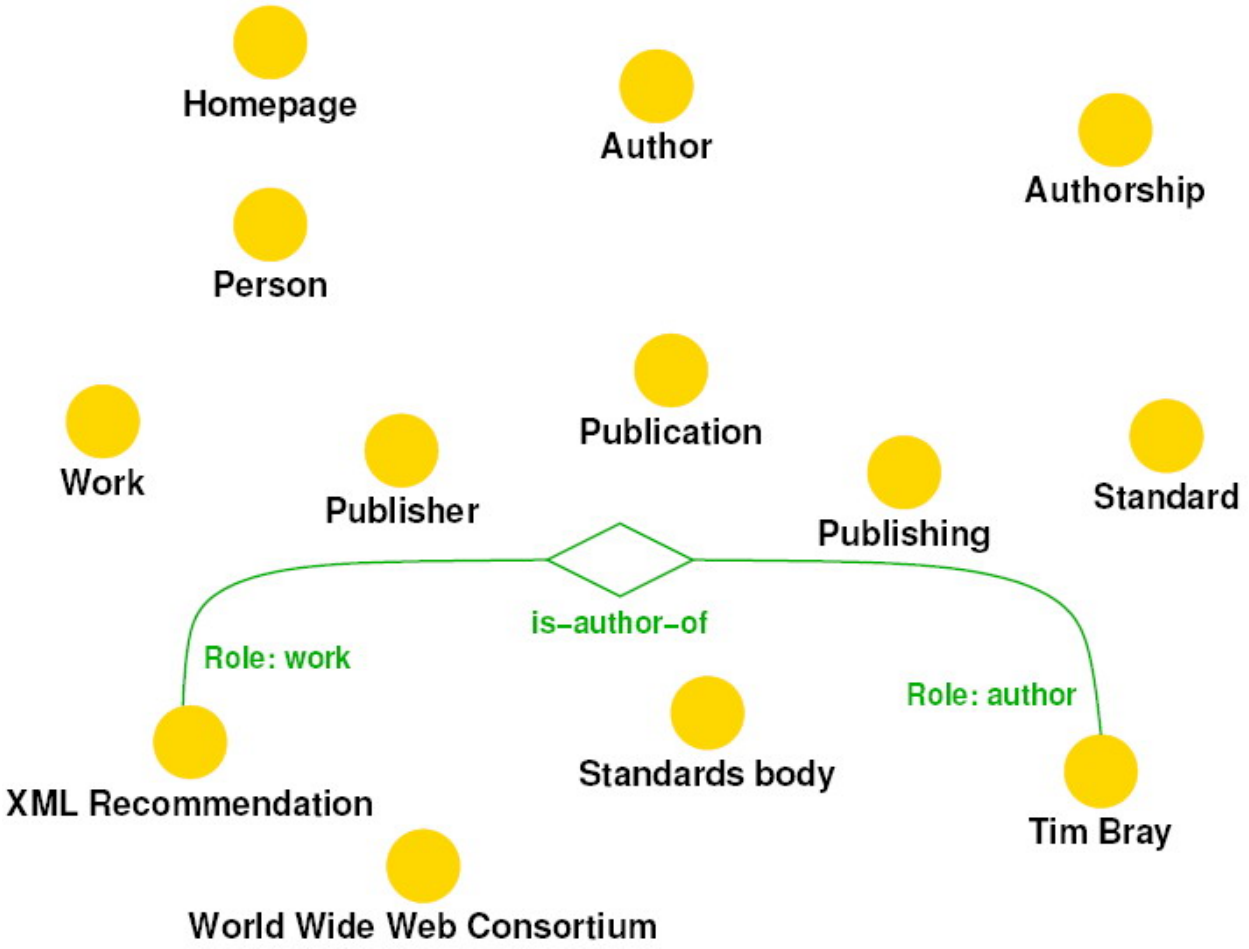
Topic Maps – Example (I)



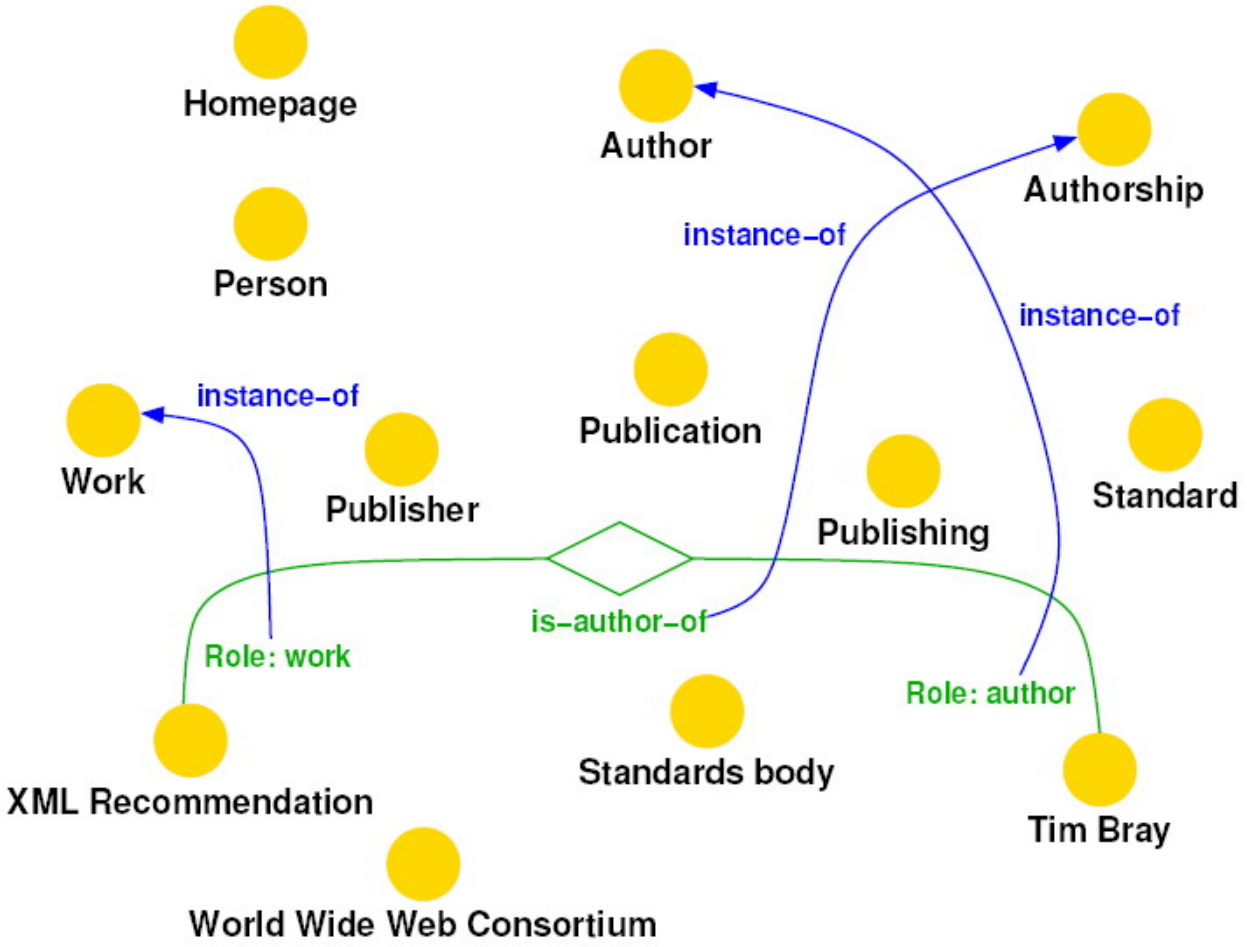
Topic Maps – Example (II)



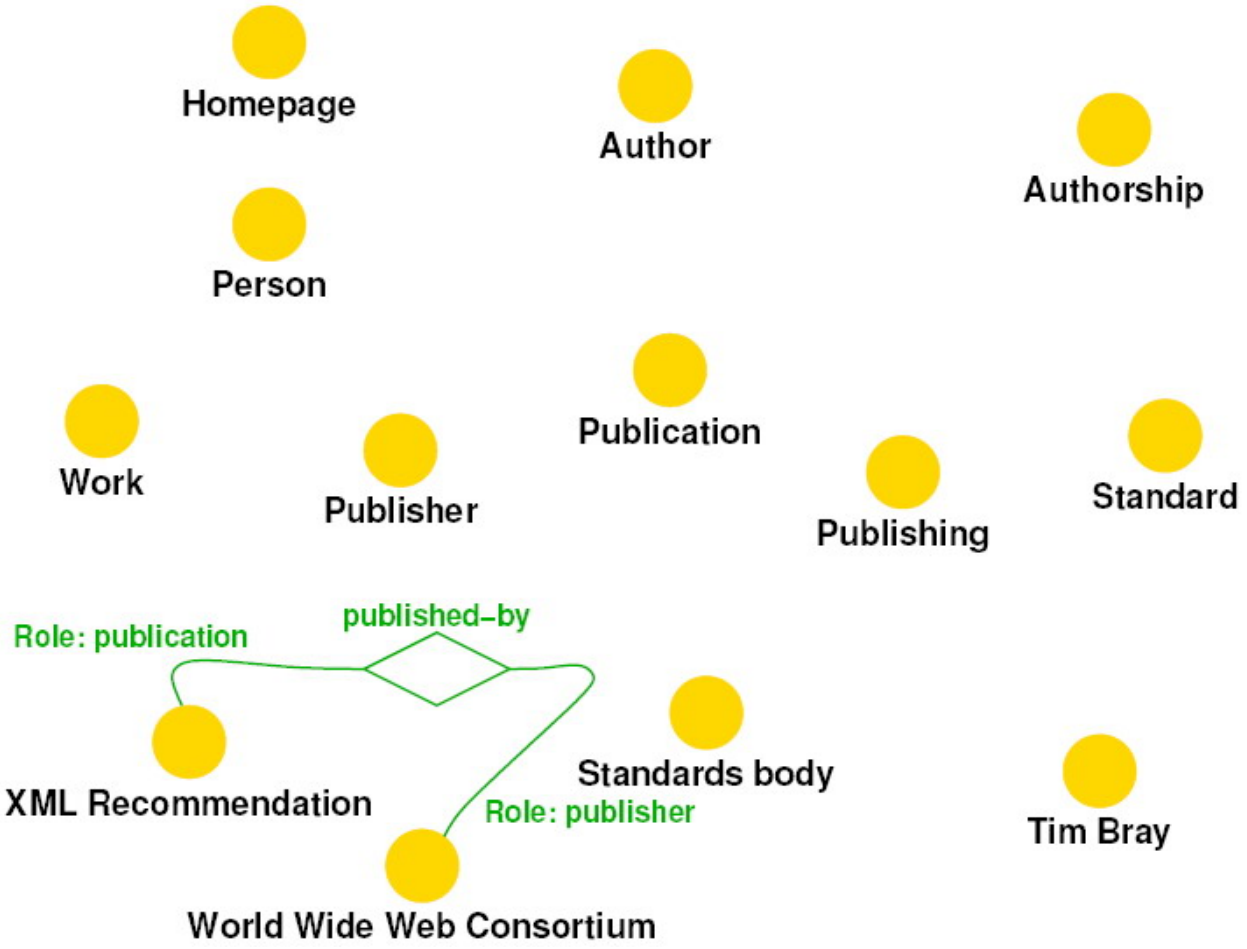
Topic Maps – Example (III)



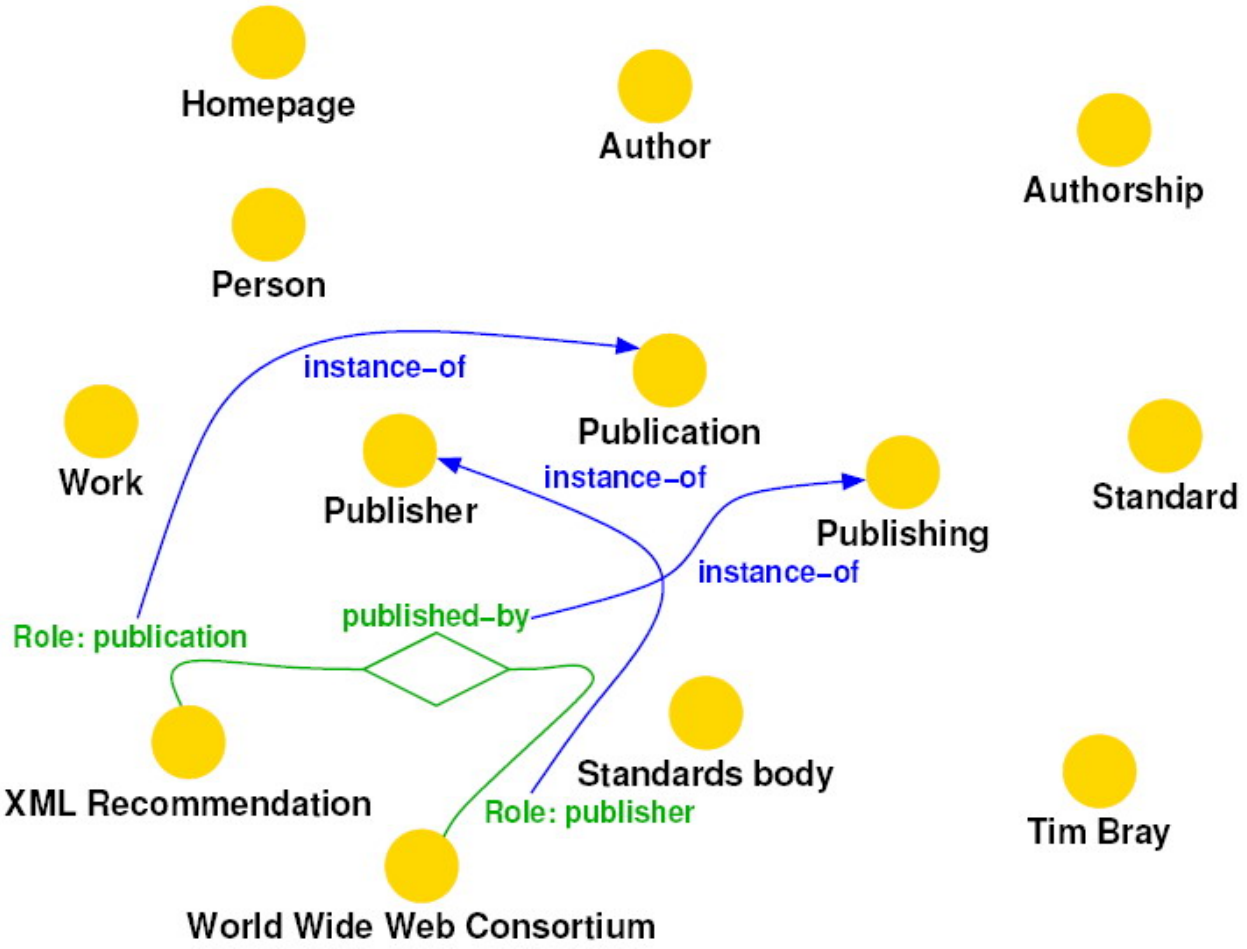
Topic Maps – Example (IV)



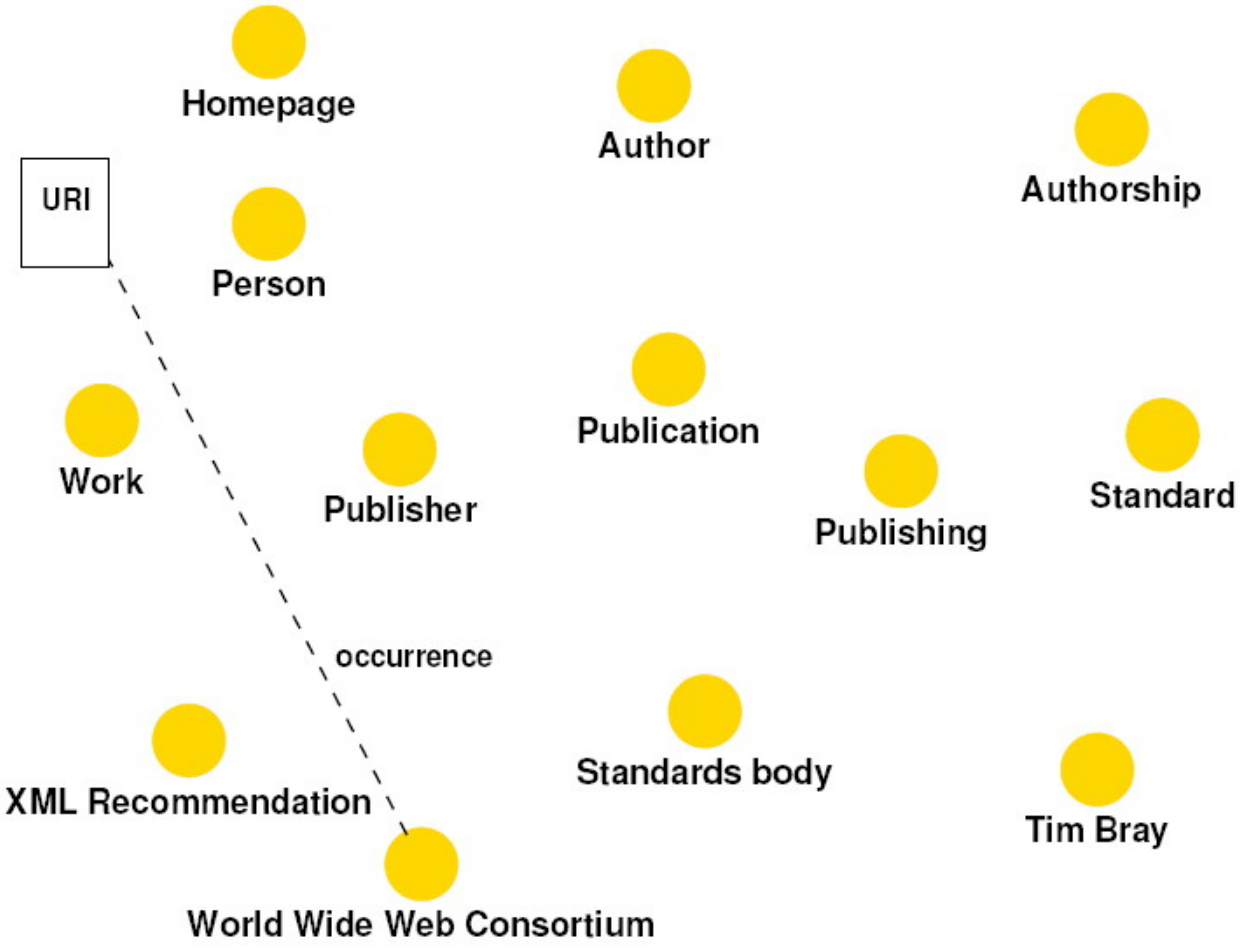
Topic Maps – Example (V)



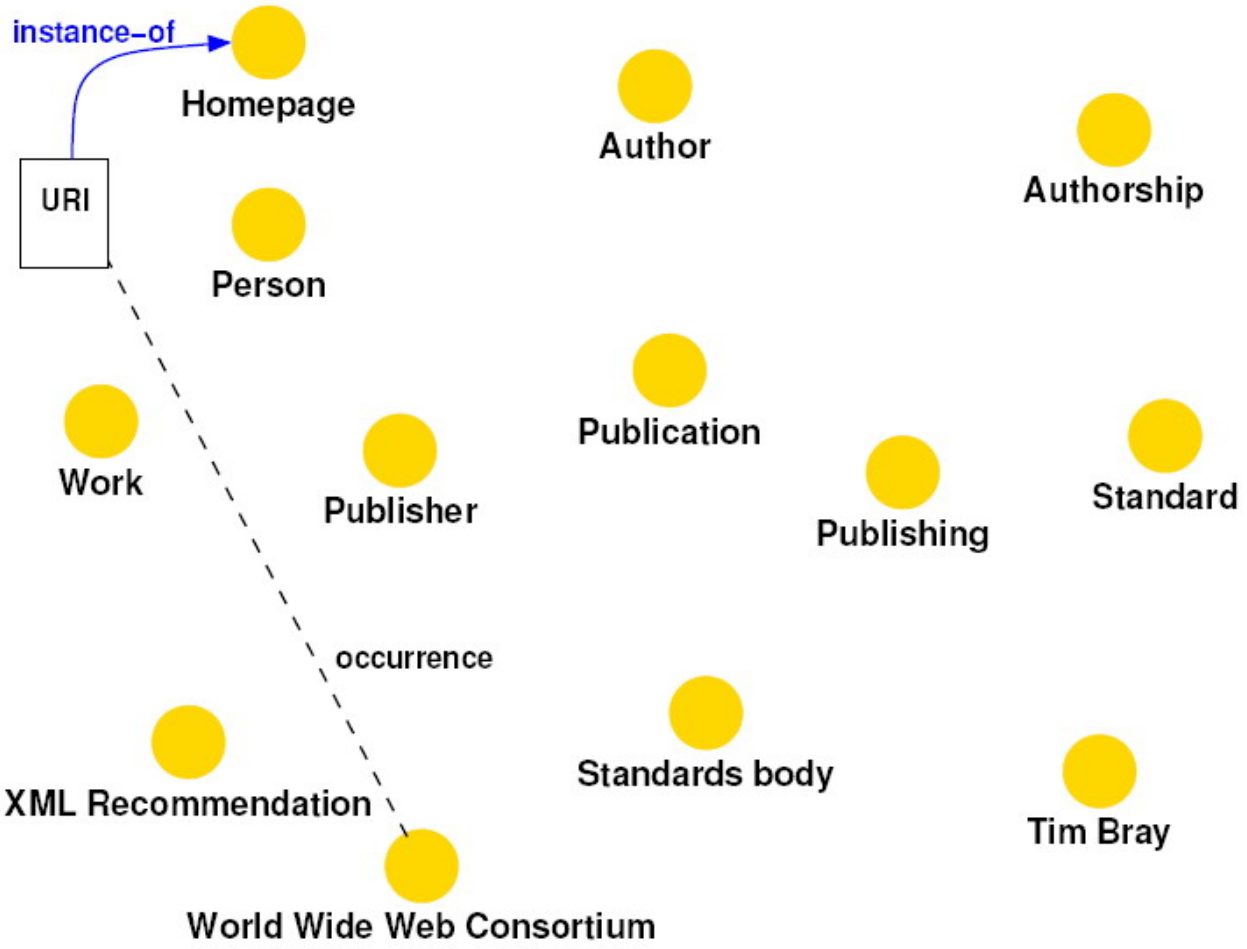
Topic Maps – Example (VI)



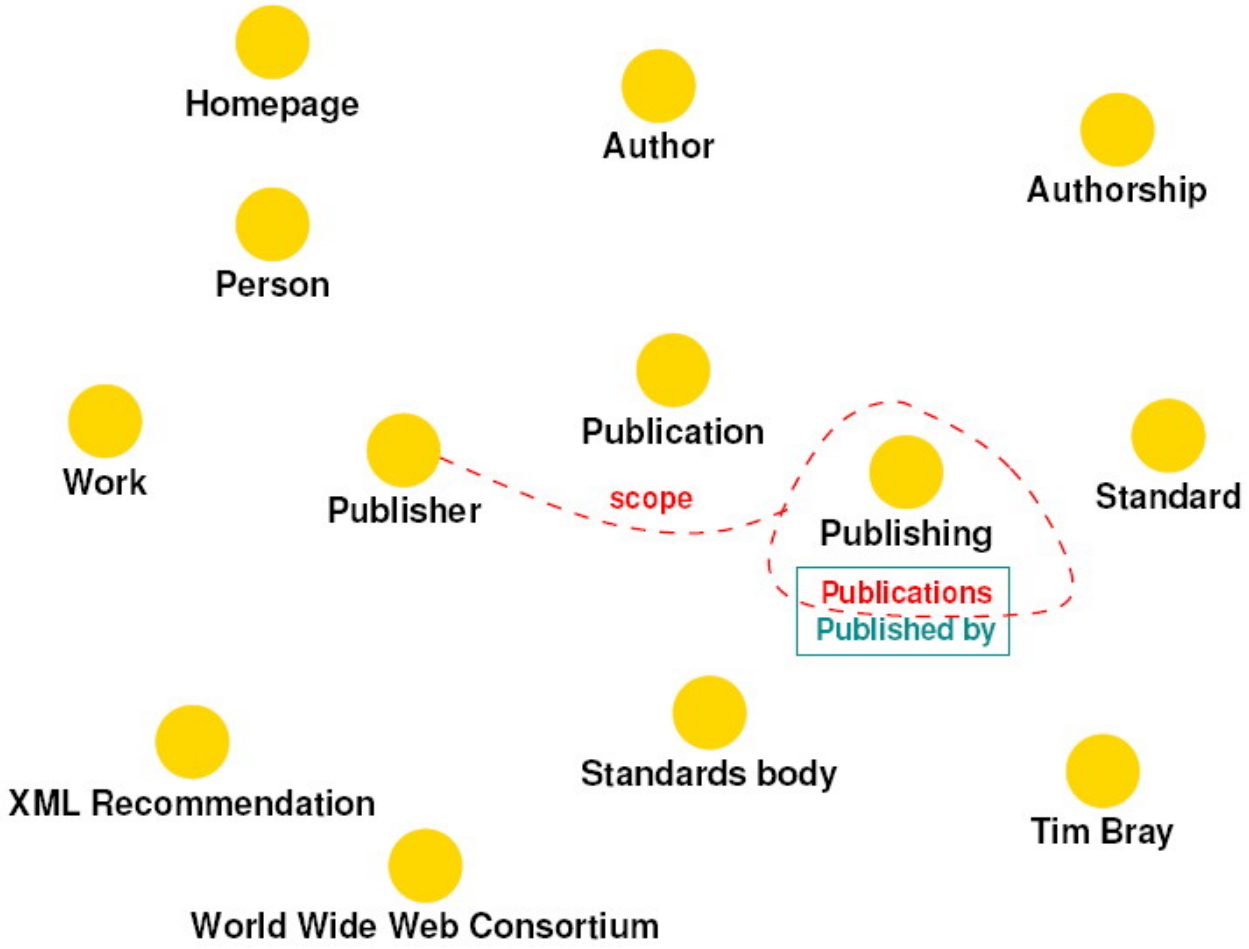
Topic Maps – Example (VII)



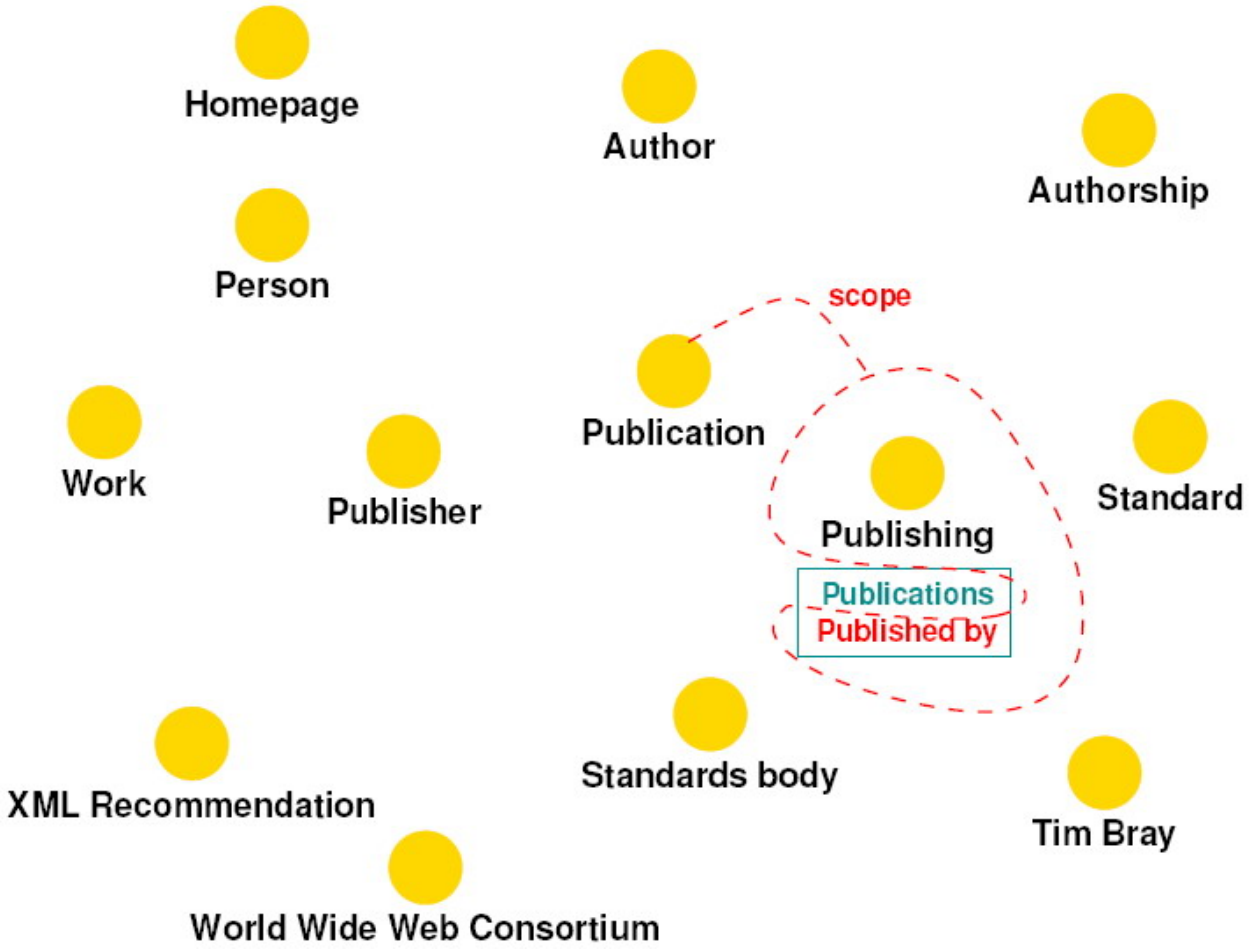
Topic Maps – Example (VIII)



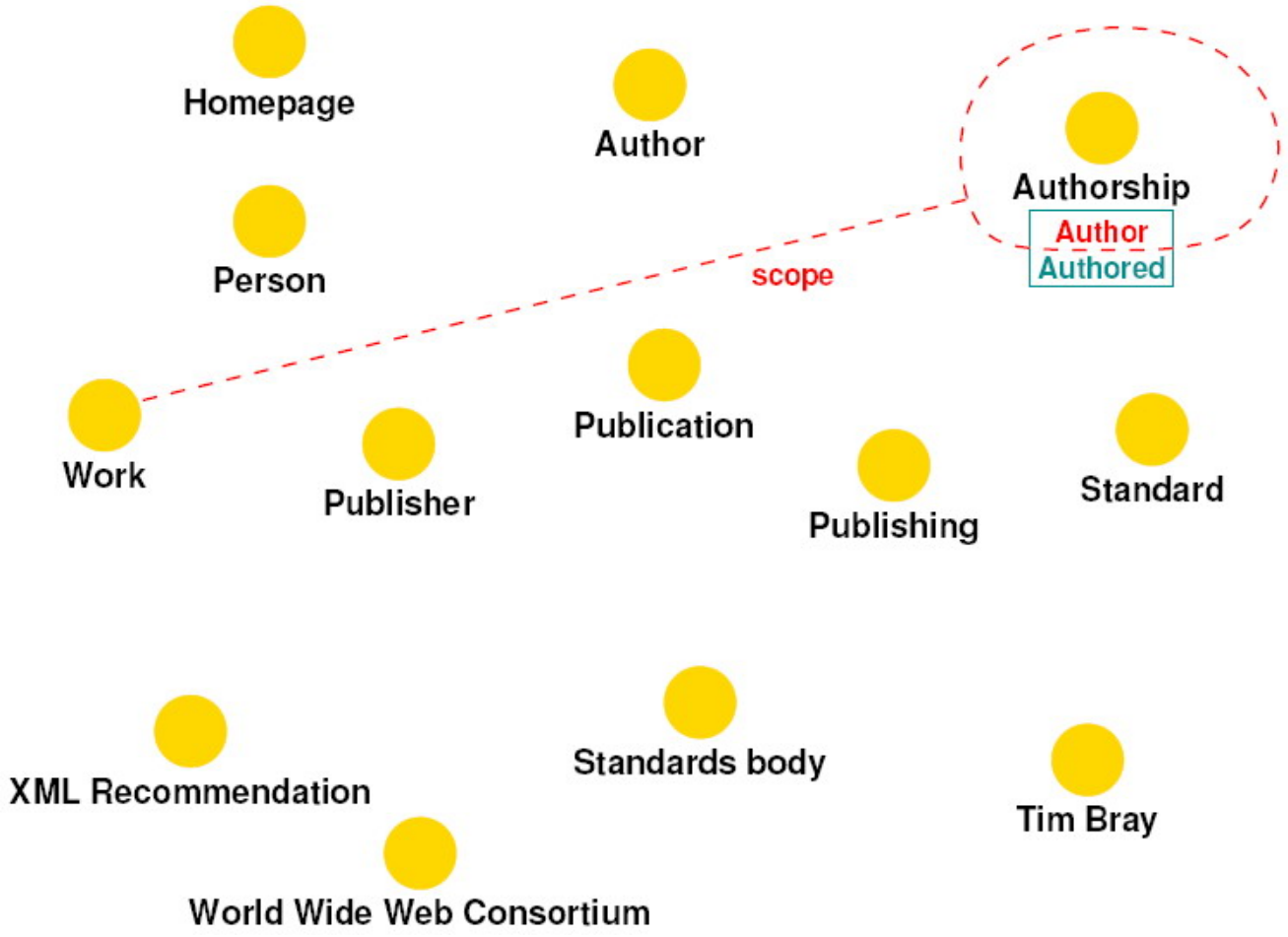
Topic Maps – Example (IX)



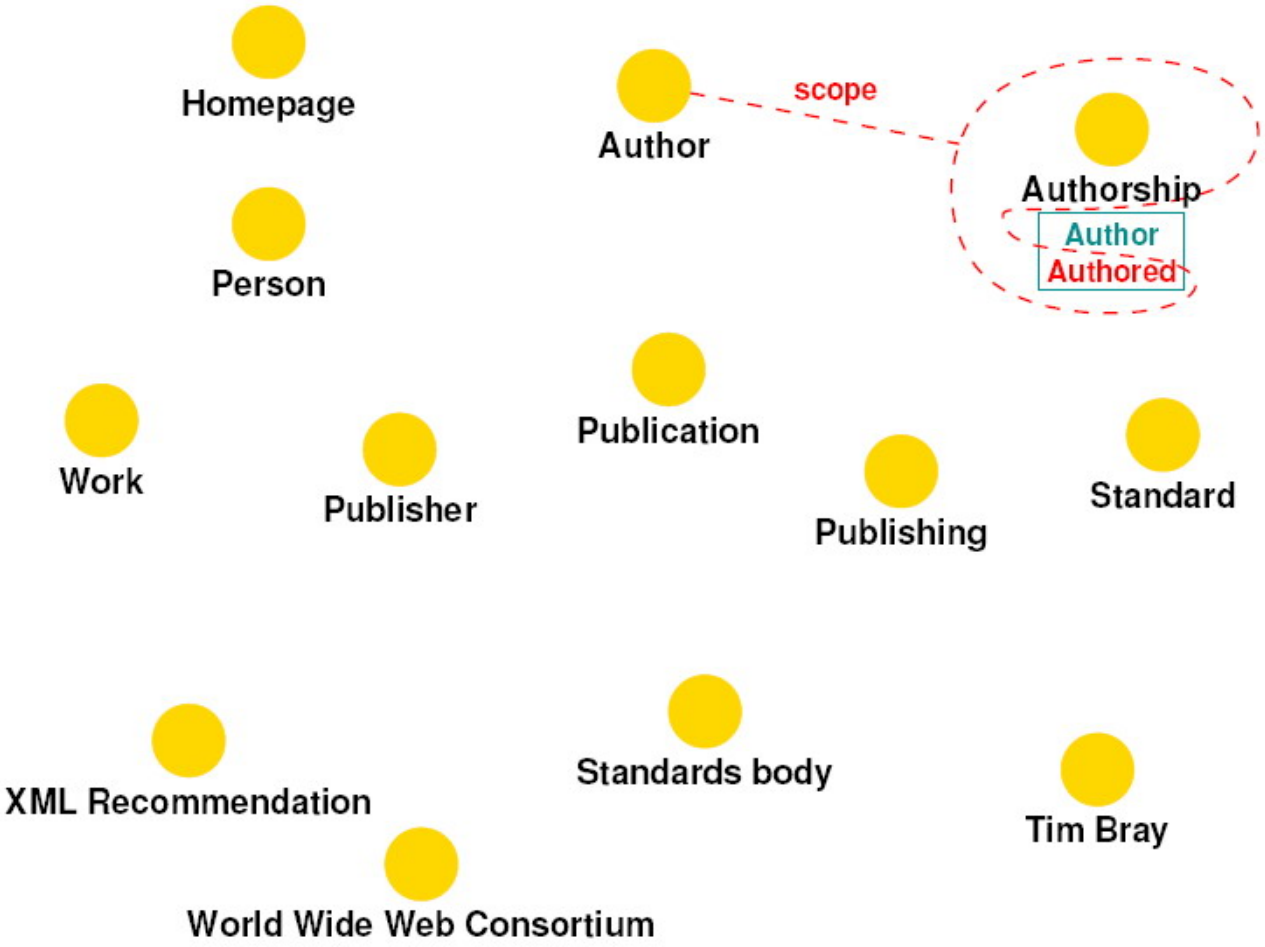
Topic Maps – Example (X)



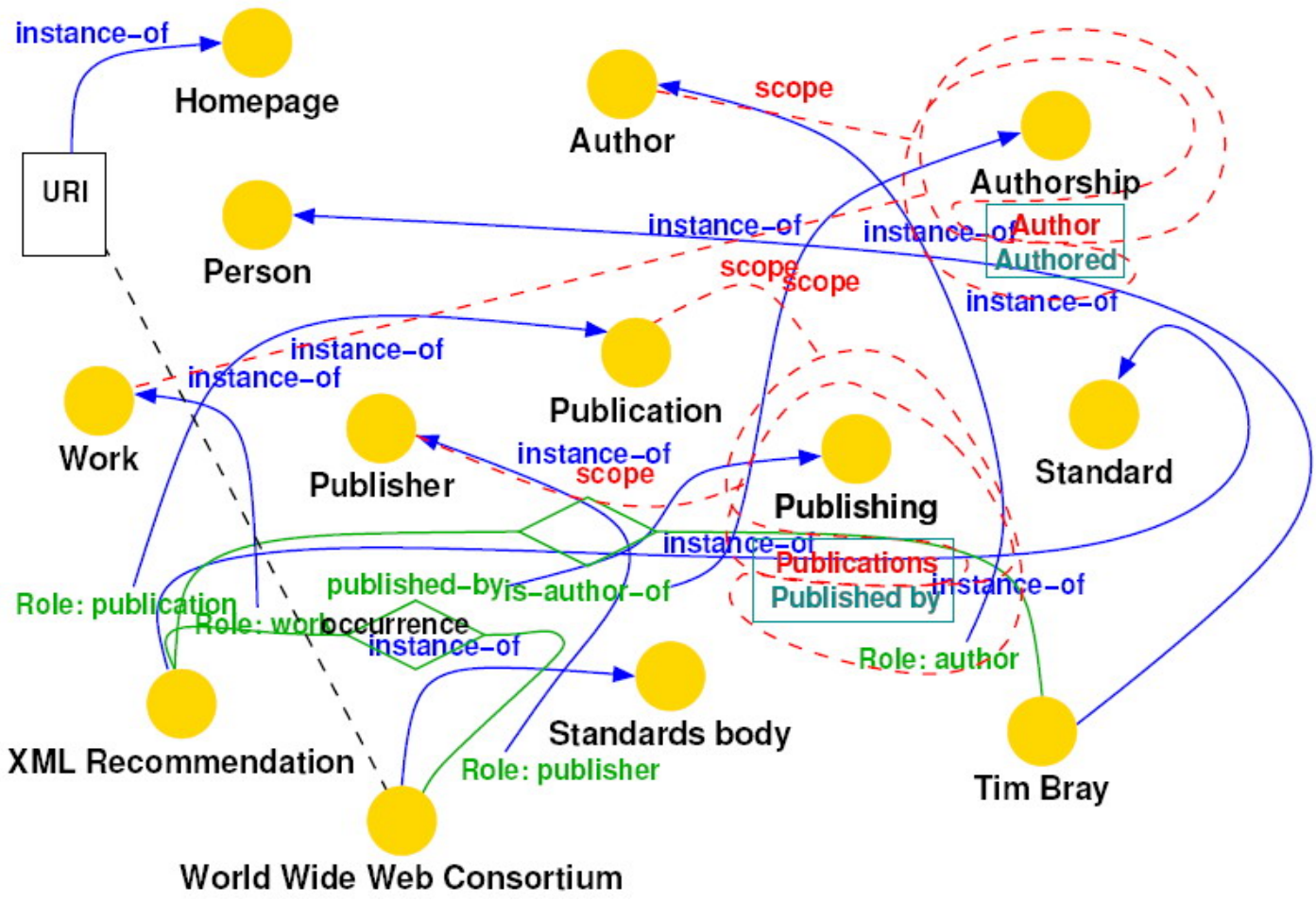
Topic Maps – Example (XI)



Topic Maps – Example (XII)



Topic Maps – Example (XIII)



XTM – XML Topic Maps

- are an abstract model and XML grammar for the exchange of web-based topic maps created by the TopicMaps.Org Authoring Group (2001)
- design goals:
  - ease of use
  - support for a wide range of applications
  - standards compatible
  - ideally no optional features
  - short and concise specification
  - XTM documents shall be easy to create, read and understand

XTM Syntax Overview (I)

- topicRef: reference to a topic element
- subjectIndicatorRef: reference to a subject indicator
- scope: reference to topic(s) that comprise the scope
- instanceOf: points to a topic representing a class
- topicMap: topic map document element
- topic: topic element
- subjectIdentity: subject reified by topic
- baseName: base name of a topic
- baseNameString: base name string container
- variant: alternate forms of base name

XTM Syntax Overview (II)

- variantName: container for variant name
- parameters: processing context for variant
- association: topic association
- member: member in topic association
- roleSpec: points to a topic serving as an association role
- occurrence: resources regarded as an occurrence
- resourceRef: reference to a resource
- resourceData: container for resource data
- mergeMap: merge with another topic map

XTM - topicRef

- Synopsis:
  - The <topicRef> element provides a URI reference to a topic. The target of a <topicRef> link must resolve to a <topic> element child of a <topicMap> document that conforms to this XTM specification. The target <topic> need not be in the document entity of origin.
- Content Model:
  - `<!ELEMENT topicRef EMPTY>`
- Attributes:
  - `<!ATTLIST topicRef`

    `id ID #IMPLIED`

    `xlink:type NMTOKEN #FIXED 'simple'`

    `xlink:href CDATA #REQUIRED>`
- Example:
  - `<topicRef xlink:href="http://www.topicmaps.org/xtm/1.0/language.xtm#en"/>`

XTM - subjectIndicatorRef

- Synopsis:
  - The <subjectIndicatorRef> element provides a URI reference to a resource that acts as a subject indicator.
- Content Model:
  - `<!ELEMENT subjectIndicatorRef EMPTY>`
- Attributes:
  - `<!ATTLIST subjectIndicatorRef`
    `id ID #IMPLIED`
    `xlink:type NMTOKEN #FIXED 'simple'`
    `xlink:href CDATA #REQUIRED>`
- Example:
  - `<subjectIndicatorRef`
    `xlink:href="http://www.shakespeare.org/plays.html#hamlet"/>`

XTM - scope

- Synopsis:
  - The <scope> element consists of one or more <topicRef>, <resourceRef>, or <subjectIndicatorRef> elements. The union of the subjects corresponding to these elements specifies the context in which the assignment of the topic characteristic is considered to be valid.
- Content Model:
  - `<!ELEMENT scope (topicRef | resourceRef | subjectIndicatorRef)+>`
- Attributes:
  - `<!ATTLIST scope`

    `id ID #IMPLIED>`
- Example:
  - `<scope>`

    `<topicRef xlink:href="#tragedy"/>`

    `<topicRef xlink:href="#theatre"/>`

    `</scope>`

XTM - instanceOf

- Synopsis:
  - The <instanceOf> element specifies the class to which its parent belongs, via a <topicRef> or <subjectIndicatorRef> child element.
- Content Model:
  - `<!ELEMENT instanceOf (topicRef | subjectIndicatorRef) >`
- Attributes:
  - `<!ATTLIST instanceOf`
    `id ID #IMPLIED>`
- Example:
  - `<topic id="hamlet">`
    `<instanceOf>`
    `<subjectIndicatorRef xlink:href="http://www.shakespeare.org/plays.html"/>`
    `</instanceOf>`
    `</topic>`

XTM – topicMap (I)

- Synopsis:
  - The <topicMap> element is the parent of all <topic>, <association>, and <mergeMap> elements in the topic map document.
- Content Model:
  - `<!ELEMENT topicMap (topic | association | mergeMap)*>`
- Attributes:
  - `<!ATTLIST topicMap`

    `id ID #IMPLIED`

    `xmlns CDATA #FIXED 'http://www.topicmaps.org/xtm/1.0/'`

    `xmlns:xlink CDATA #FIXED 'http://www.w3.org/1999/xlink'`

    `xml:base CDATA #IMPLIED>`

XTM – topicMap (II)

- Example:
  - <?xml version="1.0"?>

```
<!DOCTYPE topicMap
  PUBLIC "-//TopicMaps.Org//DTD XML Topic Map (XTM) 1.0//EN"
  "file://usr/local/home/gromit/xml/xtm/xtm1.dtd">
<topicMap xmlns='http://www.topicmaps.org/xtm/1.0/'
  xmlns:xlink='http://www.w3.org/1999/xlink'
  xml:base='http://www.shakespeare.org/hamlet/'>
  <!-- topics, associations, and merge map directives go here -->
</topicMap>
```

XTM - topic

- Synopsis:
  - The <topic> element specifies the name and occurrence characteristics of a single topic.
- Content Model:
  - `<!ELEMENT topic (instanceOf*, subjectIdentity?, (baseName | occurrence)*)>`
- Attributes:
  - `<!ATTLIST topic`
    `id ID #REQUIRED>`
- Example:
  - `<topic id="hamlet">`
    `<instanceOf>`
    `<topicRef xlink:href="#play"/>`
    `</instanceOf>`
    `<!-- base names and occurrences go here -->`
    `</topic>`

XTM - subjectIdentity

- Synopsis:
  - The <subjectIdentity> element specifies the subject that is reified by a topic, via <resourceRef>, <subjectIndicatorRef>, and/or <topicRef> child elements.
- Content Model:
  - `<!ELEMENT subjectIdentity (resourceRef?, (topicRef | subjectIndicatorRef)*)>`
- Attributes:
  - `<!ATTLIST subjectIdentity`
    `id ID #IMPLIED>`
- Example:
  - `<topic id="dk">`
    `<subjectIdentity>`
    `<subjectIndicatorRef`
    `xlink:href="http://www.topicmaps.org/xtm/1.0/country.xtm#dk"/>`
    `</subjectIdentity>`
    `</topic>`

XTM - baseName

- Synopsis:
  - The <baseName> element specifies a topic name in form of a <baseNameString> child element.
- Content Model:
  - `<!ELEMENT baseName (scope?, baseNameString, variant*)>`
- Attributes:
  - `<!ATTLIST baseName`
    `id ID #IMPLIED>`
- Example:

```
<topic id="shakespeare">
  <baseName>
    <baseNameString>William Shakespeare</baseNameString>
  </baseName>
</topic>
```

XTM – `baseNameString` (I)

- Synopsis:
  - The <baseNameString> element is a string that represents the base name of its ancestor <topic> parent.
- Content Model:
  - `<!ELEMENT baseNameString (#PCDATA)>`
- Attributes:
  - `<!ATTLIST baseNameString`

    `id ID #IMPLIED>`

XTM – baseNameString (II)

- Example:
  - <topic id="written-by">
    <baseName>
    <baseNameString>written by</baseNameString>
    </baseName>
    <baseName>
    <scope>
    <topicRef xlink:href="#author"/>
    </scope>
    <baseNameString>author of</baseNameString>
    </baseName>
    </topic>

XTM – `variant` (I)

- Synopsis:
  - The <variant> element is an alternate form of a topic's base name appropriate for a processing context specified by the variant's <parameters> child element. Among these contexts may be sorting and display.
- Content Model:
  - `<!ELEMENT variant (parameters, variantName?, variant*)>`
- Attributes:
  - `<!ATTLIST variant`

    `id ID #IMPLIED>`

XTM – variant (II)

- Example:
  - <topic id="shakespeare">
    <baseName>
    <baseNameString>William Shakespeare</baseNameString>
    <!-- form for sorting (sort name) -->
    <variant>
    <parameters>
    <topicRef xlink:href="#sort"/>
    </parameters>
    <variantName>
    <resourceData>shakespeare,william</resourceData>
    </variantName>
    </variant>
    </baseName>
    </topic>

XTM - variantName

- Synopsis:
  - The <variantName> element provides the resource to be used as a variant of a base name.
- Content Model:
  - `<!ELEMENT variantName (resourceRef | resourceData)>`
- Attributes:
  - `<!ATTLIST variantName`

    `id ID #IMPLIED>`
- Example:
  - `<variantName>`

    `<resourceData>shakespeare,william</resourceData>`

    `</variantName>`

XTM – `parameters` (I)

- Synopsis:
  - The <parameters> element consists of one or more <topicRef> or <subjectIndicatorRef> elements. The union of the subjects corresponding to these elements specifies an additional processing context in which variant names in the variant's subtree are considered to be appropriate.
- Content Model:
  - `<!ELEMENT parameters (topicRef | subjectIndicatorRef)+>`
- Attributes:
  - `<!ATTLIST parameters`

    `id ID #IMPLIED>`

XTM – parameters (II)

- Example:
  - <topic id="shakespeare">
    <baseName>
    <baseNameString>William Shakespeare</baseNameString>
    <!-- form for sorting (sort name) -->
    <variant>
    <parameters>
    <topicRef xlink:href="#sort"/>
    </parameters>
    <variantName>
    <resourceData>shakespeare,william</resourceData>
    </variantName>
    </variant>
    </baseName>
    </topic>

XTM – association (I)

- Synopsis:
  - The <association> element asserts a relationship among topics that play roles as members of the association.
- Content Model:
  - `<!ELEMENT association (instanceOf?, scope?, member+)>`
- Attributes:
  - `<!ATTLIST association id ID #IMPLIED>`

XTM – association (II)

- Example:

```
<association id="will-wrote-hamlet">
  <instanceOf>
    <topicRef xlink:href="#written-by"/>
  </instanceOf>
  <member>
    <roleSpec><topicRef xlink:href="#author"/></roleSpec>
    <topicRef xlink:href="#shakespeare"/>
  </member>
  <member>
    <roleSpec><topicRef xlink:href="#work"/></roleSpec>
    <topicRef xlink:href="#hamlet"/>
  </member>
</association>
```

XTM - member

- Synopsis:
  - The <member> element specifies all topics that play a given role in an association. The <roleSpec> element specifies the role played by these topics.
- Content Model:
  - `<!ELEMENT member (roleSpec?, (topicRef | resourceRef | subjectIndicatorRef)+)>`
- Attributes:
  - `<!ATTLIST member id ID #IMPLIED>`
- Example:

```
<member>
  <roleSpec><topicRef xlink:href="#work"/></roleSpec>
  <topicRef xlink:href="#hamlet"/>
</member>
```

XTM - roleSpec

- Synopsis:
  - The <roleSpec> element specifies the role played by a member in an association.
- Content Model:
  - `<!ELEMENT roleSpec (topicRef | subjectIndicatorRef)>`
- Attributes:
  - `<!ATTLIST roleSpec`
    `id ID #IMPLIED>`
- Example:
  - `<roleSpec>`
    `<topicRef xlink:href="#work"/>`
    `</roleSpec>`

XTM – occurence (I)

- Synopsis:
  - The <occurrence> element specifies a resource supplying information relevant to a topic.
- Content Model:
  - `<!ELEMENT occurrence (instanceOf?, scope?, (resourceRef | resourceData))>`
- Attributes:
  - `<!ATTLIST occurrence`
    `id ID #IMPLIED>`

XTM – occurence (II)

- Example:
  - <topic id="hamlet">
    <occurrence id="hamlet-in-xml">
    <instanceOf>
    <topicRef xlink:href="#xml-version"/>
    </instanceOf>
    <resourceRef
    xlink:href="http://www.uwaterloo.ca/relander/XML/hamlet.xml"/>
    </occurrence>
    </topic>

XTM – `resourceRef` (I)

- Synopsis:
  - The <resourceRef> element provides a URI reference to a resource:
    - 1. as occurrences of topics (in <occurrence> elements)
    - 2. as addressable subjects (in <member>, <mergeMap>, <scope>, and <subjectIdentity> elements)
    - 3. as variant names of topics (in <variantName> elements)
- Content Model:
  - `<!ELEMENT resourceRef EMPTY>`
- Attributes:
  - `<!ATTLIST resourceRef`

    `id ID #IMPLIED`

    `xlink:type NMTOKEN #FIXED 'simple'`

    `xlink:href CDATA #REQUIRED>`

XTM – `resourceRef` (II)

- Example:
  -
```
<occurrence id="hamlet-in-xml">
  <instanceOf>
    <topicRef xlink:href="#xml-version"/>
  </instanceOf>
  <resourceRef
xlink:href="http://www.uwaterloo.ca/relander/XML/hamlet.xml"/>
</occurrence>
```

XTM – `resourceData` (I)

- Synopsis:
  - The <resourceData> element contains information in the form of character data that may be
    - 1. an occurrence of a topic, or
    - 2. a variant form of a base name.
- Content Model:
  - `<!ELEMENT resourceData (#PCDATA)>`
- Attributes:
  - `<!ATTLIST resourceData`

    `id ID #IMPLIED>`

XTM – resourceData (II)

- Example:
  -
```
<topic id="hamlet">
  <occurrence>
    <instanceOf>
      <topicRef xlink:href="#date-of-composition"/>
    </instanceOf>
    <resourceData>1600-01</resourceData>
  </occurrence>
</topic>
```

XTM – mergeMap (I)

- Synopsis:
  - A <mergeMap> element references an external <topicMap> element through an xlink:href attribute containing a URI. It is a directive to merge the containing topic map and the referenced topic map.
- Content Model:
  - `<!ELEMENT mergeMap (topicRef | resourceRef | subjectIndicatorRef)*>`
- Attributes:
  - `<!ATTLIST mergeMap`

    `id ID #IMPLIED`

    `xlink:type NMTOKEN #FIXED 'simple'`

    `xlink:href CDATA #REQUIRED>`

XTM – mergeMap (II)

- Example:
  - <mergeMap xlink:href="http://www.shakespeare.org/plays.xtm">
    <topicRef xlink:href="#shakespeare"/>
    <topicRef xlink:href="#drama"/>
    </mergeMap>
    <mergeMap xlink:href="http://www.shakespeare.org/biography.xtm">
    <resourceRef
    xlink:href="http://www.shakespeare.org/biography.xtm"/>
    </mergeMap>

Resources

- Geroimenko, V.; Chen, C. (2005): Visualizing the Semantic Web. XML-Based Internet and Information Visualization. Springer.
- XTM 1.0 specification
  - http://topicmaps.org/xtm/index.html
- XTM 2.0 draft
  - http://www.isotopicmaps.org/sam/sam-xtm/
- Topic Map Data Model
  - http://www.isotopicmaps.org/sam/sam-model/
- Topic Map Designer – free editor and graph viewer
  - http://www.topicmap-design.com/
- more tools
  - http://topicmap.com/topicmap/tools.html

Summary

- XML
  - basics (syntax, structure, well-formedness, entities, namespaces, notations)
  - DTDs
  - XML schema
  - CSS, data binding, DOM, XSL and XSLT
- Semantic Web
  - basics, meta data, classification systems, taxonomies, thesauri
  - RDF and RDFS (RDF graph, RDF syntax, classes, resources, properties, objects)
  - Dublin Core
  - XBRL
  - Ontologies, OWL (basics, sublanguages, vocabulary)
- Topic Maps and XTM (basics, vocabulary)