Semistructured Data Management
Part 1 - Data on the Web

Semi-structured data models are the response to the requirements of flexibility of Web data management. They unify concepts from conventional structured data models, document and hypertext models, and allow ad-hoc data management without the necessity of having strict, pre-defined database schemas. In this part of the lecture we will first review the basic standards of the Web architecture, XML and RDF, which represent two flavors of semi-structured data models, then give two specific examples of how these new data models require new data processing techniques, and finally introduce the abstract model of graph databases, which captures essential properties of semi-structured data models and allows us to study the notion of schemas and and fundamental algorithms for processing data and schemas in a formal setting.

Today's Questions

1. The Web of Data
2. XML (Extensible Markup Language)
3. The Semantic Web
4. RDF (Resource Description Framwork)
5. Ontology Languages

What Do You Think ?

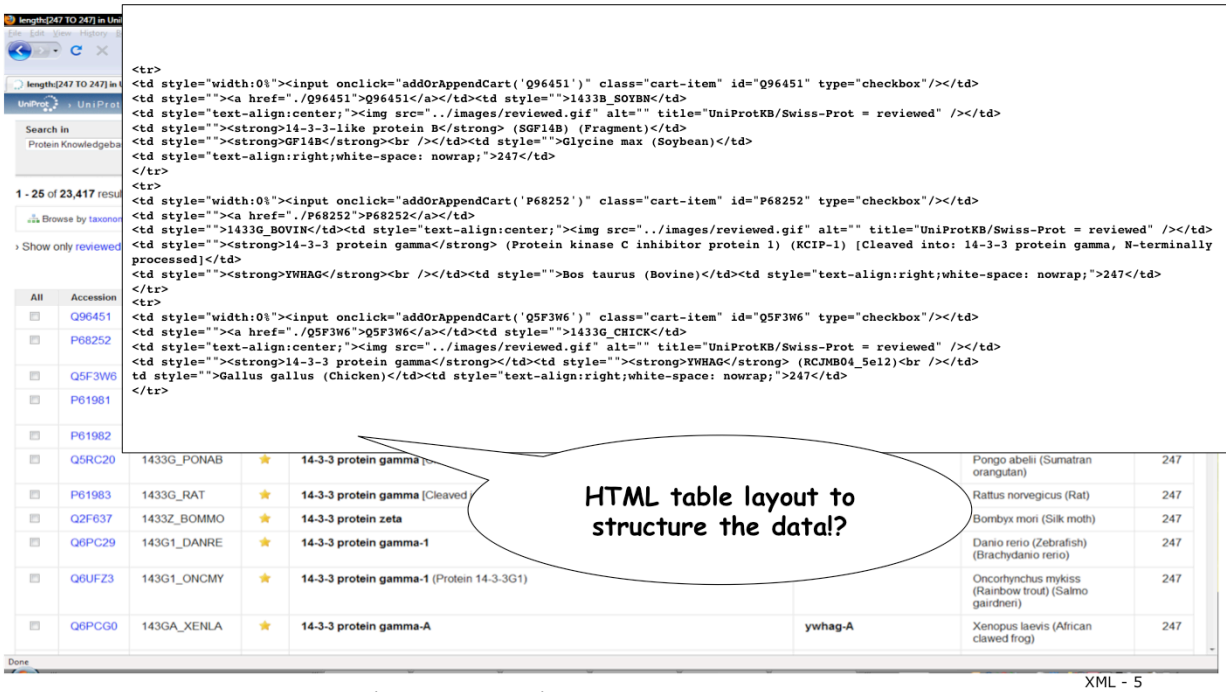
- What is XML?
- What is XML used for?

1. The Web of Data

- Example: Searching biological data
  - Based on textual content of Web pages (e.g. Google)
- Searching for data on "anglerfish"
  - Results will be precise
- This seems easy, but the same for "leech"
  - Organism leech
  - Software: LeechFTP
  - Rock band: Leech
  - Authors: Leech
  - Protein sequences: …MNTS**LEECH**MPKGD…
- Search for "257" …

This example illustrates that without providing context, as it is the case for today's text-based search techniques on the Web, it can be very difficult or impossible to interpret data correctly.

Using HTML to Structure Data



A typical approach to provide the correct of interpretation of data is to structure it. On the Web this is frequently done by using HTML (tables or lists) to present data on the Web in a structured format.

Application-specific Markup

- Limitations of HTML
  - Structure of data expressed as layout: `<tr><td> leech </td><tr>`
  - Semantics of data hard to analyse and difficult to share
  - No schemas, no constraints

- Making the meaning of data explicit
  - SwissProt: `<Species> leech </Species>`
  - EMBLChange: `<Organism> leech </Organism>`

⇒ **semi-structured data**
  - Data that contains tags, markup or similar to identify the semantics of data values and to relate different data values (e.g. hierarchically)
  - Predefined structure, e.g. schema, not required

- Examples of semi-structured data
  - Email, Microformats (e.g. hCard), XML (Extensible Markup Language)

However, using HTML markup which is intended to describe the layout of a Web page (and not the meaning of data) can be very problematic, in particular for the automated processing of the data. Even if the meaning of the data can be inferred from an HTML page, for example by analyzing the first row of data table for identifying attribute names, this processing can be very involved and error prone.

Therefore a better approach is to provide the meaning of data explicitly. For data that is published in text format (as it is the case for Web data) this can be achieved by directly embedding into the data markup (or tags) that identify the meaning. This approach is widely used, in fact already since long, e.g. for email. On the Web we find this approach with microformats that exploit HTML to embed application specific tags into the Web content (e.g. using the class attribute available in HTML). A more systematic approach to this is pursued with XML, which provides a syntactic framework for embedding application-specific markup into text documents.

These different approaches to structure data and provide meaning to data are based on specific kinds of data models. Since these data models are less strict in terms of predefining which structural elements (data types) are used in an application, these data models are also called semi-structured data models. One particularly relevant example of a semi-structured data model is XML, which has become the standard syntax for Web data and which we will explore in more detail in the following.

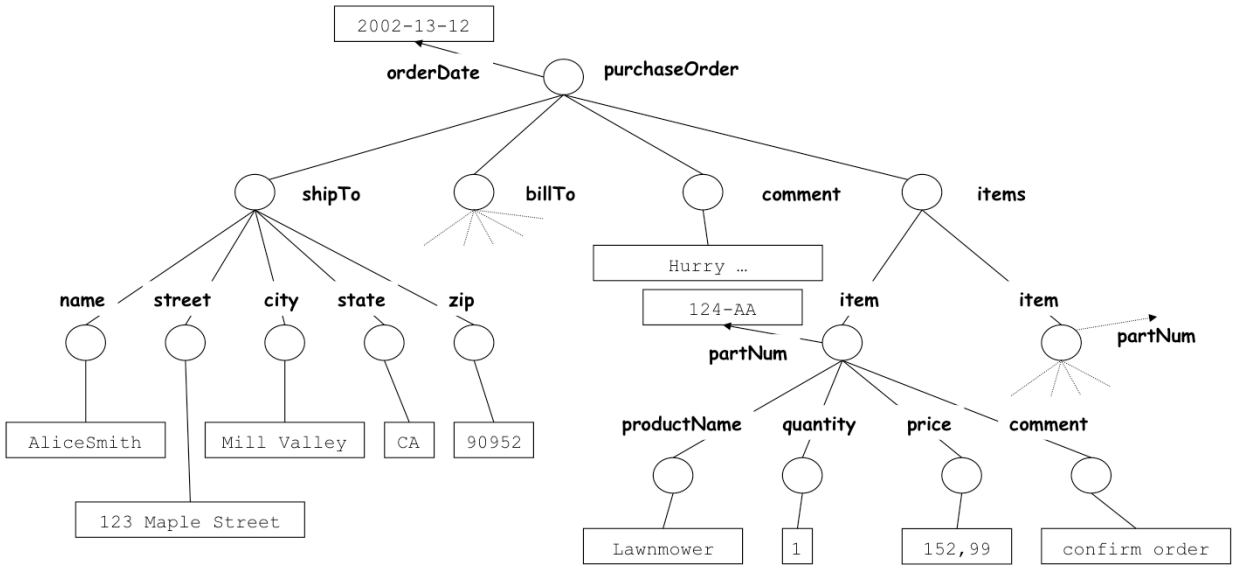
XML – as Document Language

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

XML - 7

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

XML- as Semi-structured Data Model



From the hierarchical structure of the markup one can derive a tree representation of an XML document. If we consider this tree as a data structure, we adopt the view of XML being a (semi-structured) data model.

Data and Documents

- Serialization

Document = medium for exchange of information

Information system 1

Communication

Information system 2

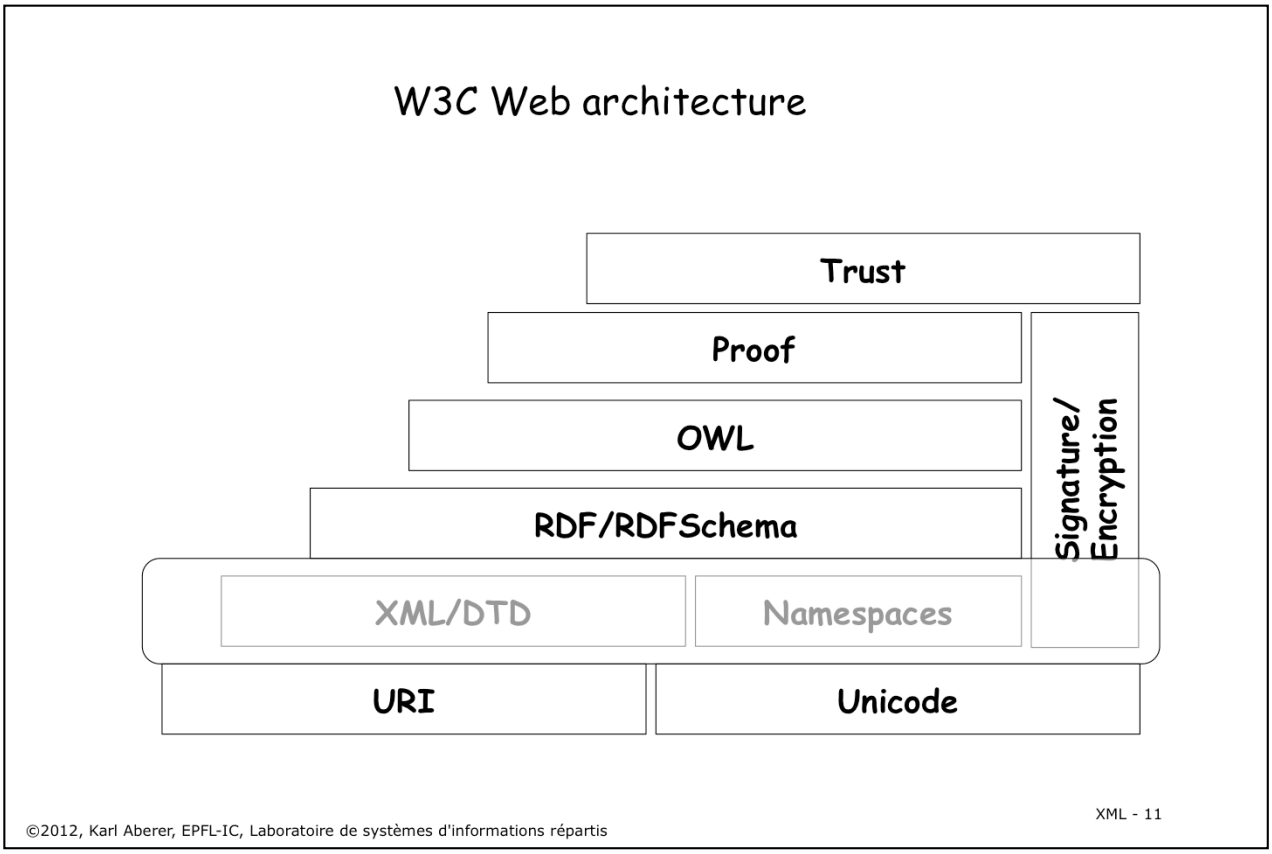
XML - 9

The duality of XML between being a document model and a data model makes it particularly suitable for applications that have to exchange data over communication networks. For every data collection or database that is represented in XML there exists a canonical encoding into a text (document). The encoding is called serialization. Since documents are sequences of symbols they can be naturally exchanged over communication networks.

What is XML ?

- Interpretation depends on the viewpoint and intended use
  - a language to describe the structure of documents.
  - foundation of the W3C architecture for the Web.
  - the successor of HTML.
  - a method to represent structured data within text documents.
  - a standard data exchange format.
  - a data model for **semi-structured data**
- XML for Web data management
  - XML as data model for managing semi-structured data
  - XML as canonical model to syntactically integrate heterogeneous data
  - XML as canonical data format to exchange data among information systems

Indeed there exist multiple perspectives on what XML is: As stated earlier XML can be considered as a document model. It has been derived from SGML, a very similar though more complex document model, that has been widely used in the publishing industry for document processing. From the perspective of the W3C, XML is part of the foundation of its Web architecture (see also next slide). Since XML overcomes the limitations of expressing semantics in HTML it is often also considered as a successor of HTML. XML is also considered as a data exchange format, respectively as a language to define domain-specific data exchange formats. As such it is adopted in many domains, e.g. for defining Electronic Data exchange Formats (EDI). And as shown before, from a data management perspective XML can also be considered as a semi-structured data model.



This figure displays the stack of architectural layers of the Web. At the syntactic level URIs provide a reference mechanism and Unicode a uniform encoding of textual content. On top of this XML provides the framework for structuring information. Namespaces are an important element to provide context-dependent unambiguous names. On top of this syntactic layers we find the framework for dealing with semantics, which is called the Semantic Web. The top layers of Proof (Reasoning and Trust) are rather conceptual than corresponding to concrete specifications.

Question

- The distinctive feature of semi-structured data is that
  - Schema information is embedded directly into the data
  - That it can be serialized
  - That it uses XML tags to identify the semantics of data

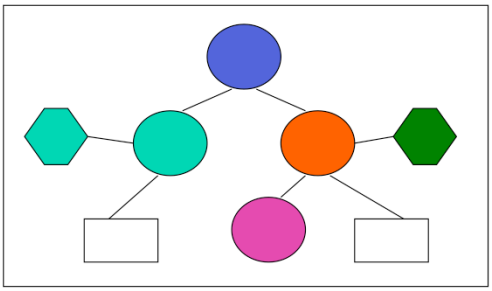
What Do You Think ?

- What is the difference between well-formed and valid XML?
  - Can you verify that an XML document is correct?
  - Can you write a correct XML document?
- What is a document type definition?
  - Can you write a document type definition?
- What is an XML Namespace?
- What is XPath?
- What is XQuery?
  - Can you write an XQuery?

2. XML Syntax

- ***Well-formed XML*** conforms to a basic XML syntax and some semantic constraints for well-formedness

- Main concepts
  - Elements: used to structure the document, identify a portion of the document
  - Attributes: associate data values with elements, used to reduce the number of elements and for typed data
  - Character data (PCDATA): textual content of the document



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


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

Well-Formed XML Syntax

- Syntax (excerpt)

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

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

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

Well-formed XML can be specified by means of a formal syntax of which we give only the basic and most important production rules. In particular one can see the hierarchical construction for the element structure. This syntax implies some of the properties that well-formed XML documents must satisfy. However, there exist additional requirements, so-called semantic constraints, on well-formed XML, which are not enforced by the grammar.

Well-Formed XML Syntax

- Syntax (excerpt)

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

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

non-XML:

```
<1-shipTo>
  Alice
</1-shipTo>
```

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

This document is not XML, since the tag name does not start with a letter.

Well-Formed XML Syntax

- Syntax (excerpt)

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

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

non-XML:

```
<shipTo>
  Alice
</shipTo>
<billTo>
  Bob
</billTo>
```

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

The example on the left is not well-formed XML, as the document has no root. Such (incomplete) documents are usually called XML fragments, provided they follow the other constraints for well-formed XML.

Well-Formed XML Syntax

- Syntax (excerpt)

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

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

non-XML:

```
<shipTo>
  Alice
</billTo>
<billTo>
  Bob
</shipTo>
```

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

This fragment violates the requirement that names of start and end tags must match.

Well-Formed XML Syntax

- Syntax (excerpt)

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

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

non-XML:

```
<shipTo
  country="US"
  country="CH"/>
```

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

This element violates the requirement that attribute names are unique.

Well-Formed XML Syntax

- Syntax (excerpt)

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

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

XML:
<shipTo country="US"/>

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

This is an example for an empty element, i.e., an element that includes no textual content.

Well-Formed XML Syntax

- Syntax (excerpt)

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

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

non-XML:

```
<shipTo>
  Alice
<billTo>
</shipTo>
  Bob
</billTo>
```

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

This fragment violates the requirement of proper nesting of tags.

Question

- The XML document below violates how many syntactic properties?
  1. 1
  2. 2
  3. 3

```
<?xml version="1.0"?>
<Order>

<Date>2003/07/04</Date>
        <CustomerId>123</CustomerName>
        <CustomerName>Acme Alpha</CustomerId>

<Item>
        <ItemId> 987</ItemId>
        <ItemName>Coupler</ItemName>
        <Quantity>5</Quantity>
</Item>

<Item/ Quantity= "5">

</Order>
```

XML Document Type Definitions (DTDs)

- Definitions
  - Definition of element and attribute names
- Content Model (regular expressions)
  - Association of attributes with elements
  - Association of elements with other elements (containment)
  - Order and cardinality constraints
- A ***valid XML document*** must satisfy the type definition given by the DTD

XML permits but does not require the use of schemas. In the XML standard, schemas are specified through so-called document type definitions DTDs). These specify

- which element and attribute names are allowed

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

The mechanism to define these schemas differs from the one used, for example, in relational databases, where complex types are used. DTDs provide a grammar that is used to produce (respectively verify) the structure of XML documents that conform to the schema. If an XML document satisfies a DTD it is called a valid XML document (with respect to the given DTD).

More recently an alternative language for defining the structure of XML documents has been introduced: XML Schema. The main difference is that XML Schemas are represented within XML and that typical consistency constraints as they are known from relational database schemas are supported in addition.

Element Declarations

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

```
<example><a/></example>
```

```
<example>example</example>
```

```
<example/>
```

```
<example>
 <a/>
 example
</example>
```

XML - 24

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

Element Declarations

- Sequence

```
<!ELEMENT example ( a, b )>
```

```
<example><a/><b/></example>
```

- Alternative

```
<!ELEMENT example ( a | b )>
```

```
<example><a/></example>
<example><b/></example>
```

- Optional (zero or one)

```
<!ELEMENT example ( a )?>
```

```
<example></example>
```

- Optional and repeatable (zero or more)

```
<!ELEMENT example ( a )*>
```

```
<example><a/><a/><a/></example>
```

- Required and repeatable (one or more)

```
<!ELEMENT example ( a )+>
```

Mixed content

```
<!ELEMENT example (#PCDATA | a)*>
```

```
<example>
 <a/>
 example
</example>
```

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

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

Attribute Declarations

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

- Example:
  Document Type Definition

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

  Document

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


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

Attribute Declarations - Types

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

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

Usage of DTDs

External DTD Declaration

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

Internal DTD Declaration

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


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

Question

- Which is the right content model for the document below?
  1. **<!ELEMENT Item (ItemId, ItemName, Quantity)+ >**
  2. **<!ELEMENT Item (Quantity | ItemName | ItemId)+ >**
  3. **<!ELEMENT Item (ItemId+ , ItemName+ , Quantity+) >**

```
<?xml version="1.0"?>
<Order>

<Item>
        <ItemId> 987</ItemId>
        <ItemName>Coupler</ItemName>
        <Quantity>5</Quantity>
</Item>

<Item>
        <ItemId> 321</ItemId>
        <ItemName>Wheel</ItemName>
        <ItemName>Special Size</ItemName>
</Item>

</Order>
```

XML Namespaces

- An **XML namespace** is a collection of names (markup vocabulary)
  - identified by a URI reference
  - agreed element and attribute names

```
<rdf:RDF xml:lang="en"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"     xmlns:rdfs="http://
www.w3.org/2000/01/rdf-schema#">
```

- URIs are used just as unique identifiers, nothing else
  - in particular they do not refer to a DTD or schema
- Uses
  - universally agreed names
  - combination of names from different DTDs without name conflicts
  - but not combination of different DTDs

XML namespaces are a convention to distinguish different vocabularies for different applications in order to avoid names collisions. Consider just of how many XML applications would use the element name "name" and what happens if an application encounters this element name in documents originating from different sources without being able to distinguish the origin of the name. Therefore element and attribute names are prefixed by a label that is unique for a namespace. The problem is of course of how to obtain these unique labels. The solution is very simple: one uses Universal Resource Identifiers (typically URLs). But one must be careful here: the only purpose of using a URI (URL) is to have a unique prefix. There is nothing else associated with the URL, in particular the corresponding URL may not exist, it does not contain any information like a DTD or a Schema (though often useful information pertaining to the namespace is found under its specific URL).

Namespaces are important in order to provide universally agreed names with unambiguous semantics that can be exploited by applications. They also allow to combine element names from different DTDs which would otherwise be in conflict which each other. But the namespace concept is not related to the problem of combining definitions from different DTDs in any way.

Declaration

- Declaration of
  - Default namespace: xmlns (all names without prefix are supposed to be from there)
  - Identification by prefix: xmlns:ns (ns is the prefix)
- Can be declared in different positions
  - In internal DTD by using default attributes
  - In document by using attributes

```
<?xml version= "1.0"?>
<!-- element names without prefix belong to "books" -->
 <book xmlns='urn:loc.gov:book'
       xmlns:isbn= 'urn:ISBN.0-395-36341-6'>
   <title>XML Handbook</title>
   <isbn:number>1591240349</isbn:number>
 </book>
```

```
<!DOCTYPE doc [
<!ELEMENT doc (x)>
<!ELEMENT x EMPTY>
<!ATTLIST x xmlns CDATA #FIXED "http://www.jclark.com/"> ]>
<doc><x/></doc>
```

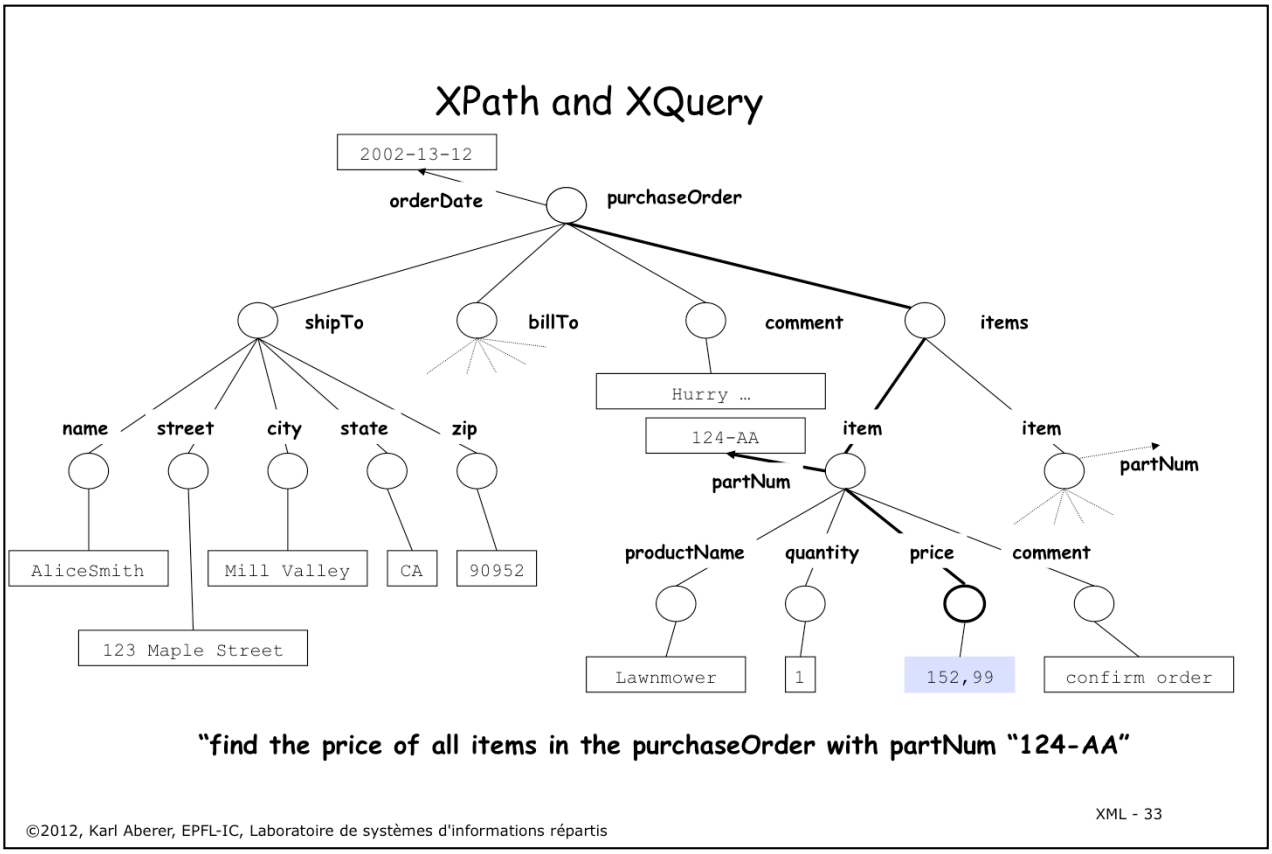

Within the scope of one element one can always declare one default namespace, to which then all the names belong that do not have a prefix. In addition, an arbitrary number of other namespaces can be distinguished by a prefix. The namespace declaration can be given both in the DTD, then it is a default attribute, or within the document by using attributes. We see in the example all the cases illustrated:

In the example document an attribute xmlns is used to declare the default namespace, such that for example the element title is from the namespace "urn:loc.gov:bok" and a second namespace isbn is declared.

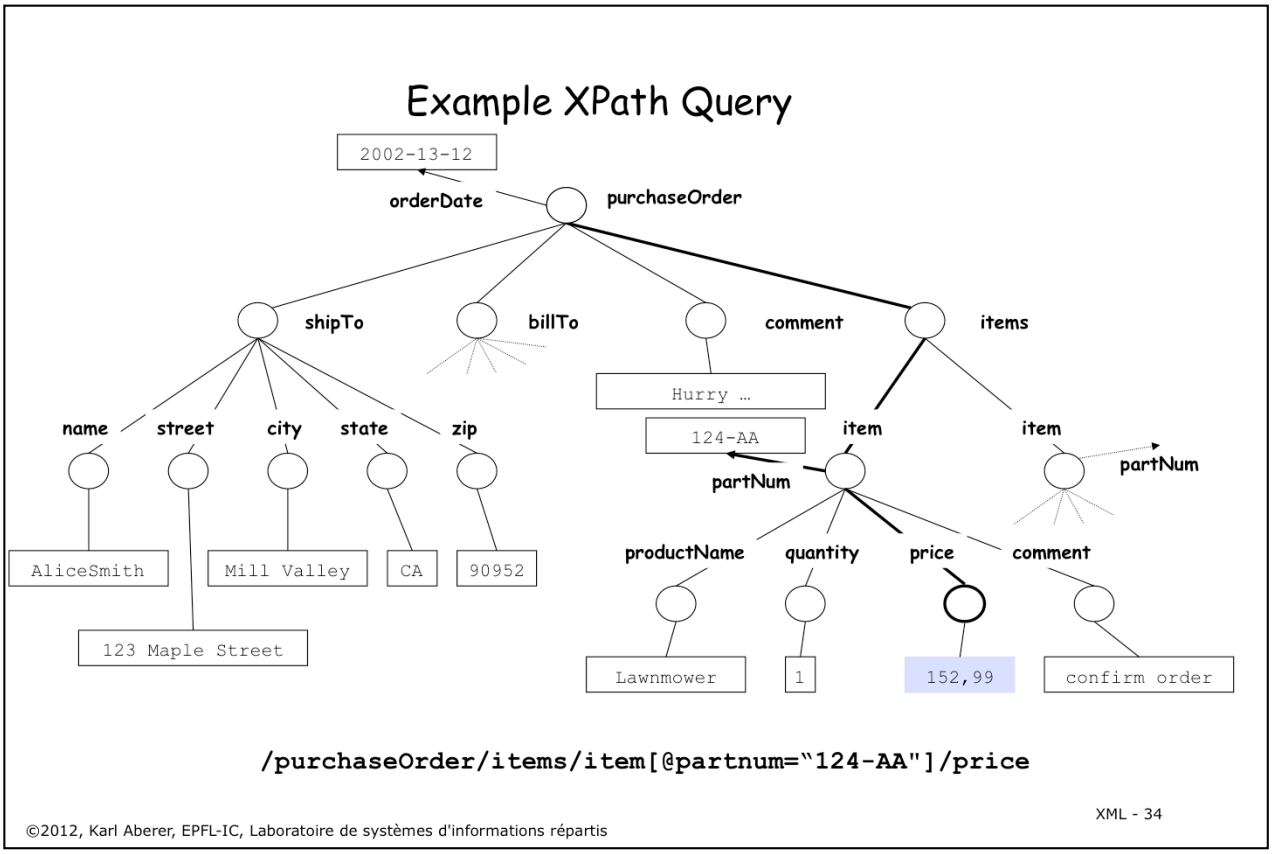
In the example DTD the element type x is equipped with the default namespace http://www.jclark.com/. Note that the keyword FIXED is used in the attribute declaration in order to express that it is an unchangeable attribute value.

Question

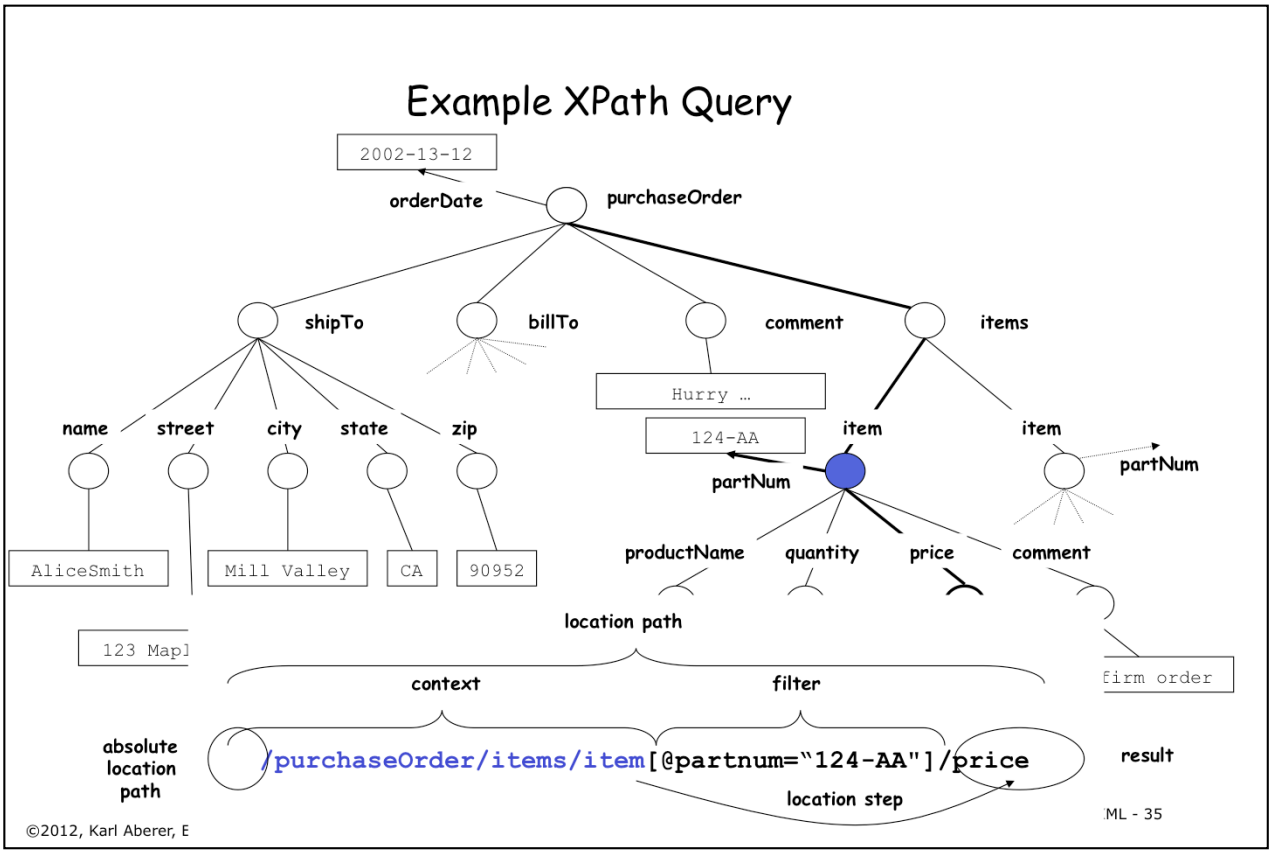
- Which is wrong? Two element names from two different namespaces
  1. Can be identical
  2. Can be used in the same document
  3. Can be used in the same document type definition
  4. Can be identified by the same URI reference



Since XML can be considered as a data model it is natural that a language for accessing and querying XML data is required. As XML data is tree-structured an important mechanism to access this data is the navigation within the tree. To that end a language for XML access based on paths has been proposed, Xpath. This example shows of how a simple query on an XML tree is evaluated …



… and of how this access is expressed in Xpath.



Xpath consists essentially of navigation operators, indicated by /, and filter expressions within square brackets that express conditions that are evaluated during navigation.

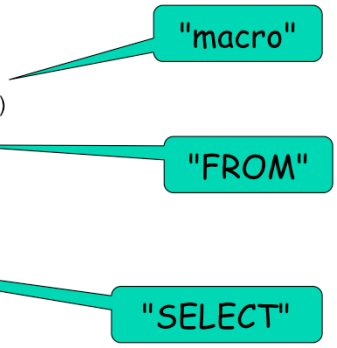
XQuery - Querying XML Data

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

Xpath lacks basic capabilities one would expect from a (declarative, set-oriented) database querying language. In particular it has no general support for set operators, it allows to return only element and attribute sets and is thus not closed (i.e. cannot produce all expressions that it uses as inputs) and it has no support of an operation that is equivalent to a relational join, which would be required to establish value-based relationships among XML document parts. To this end a query language for XML has been developed, which is called Xquery and unifies the navigation concepts introduced in Xpath with the concepts for set-oriented query processing known from XML.

Element Constructor and FLWR Expressions

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



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

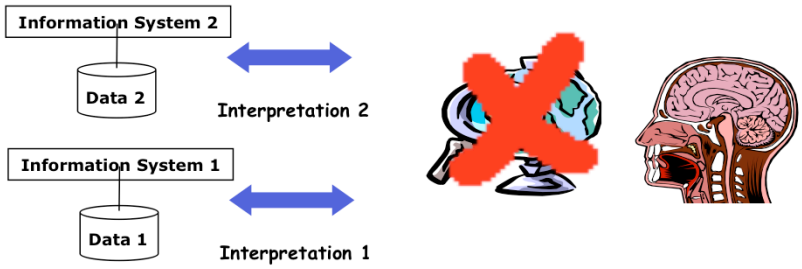
This query illustrates many of the concepts found in XQuery. Most notably one can see that XQuery allows variable binding as SQL. Different to SQL where relations are available to bind variables in XQuery they have to bind to sets that result from other XQuery expressions (this is the only possibility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

Question

- A query language is more expressive than another if after some (trivial) transformation of the data model (SQL -> XML) it can represent all queries of the less expressive language. Which is true?
  1. XQuery is more expressive than Xpath
  2. XPath is more expressive than SQL
  3. SQL is more expressive than XQuery

3. The Semantic Web

- User-defined markup (schemas): provides possibility to share interpretation of data across various applications
- Different databases – Different schemas
  - SwissProt: Find `<Species> leech </Species>`
  - EMBLChange: Find `<Organism> leech </Organism>`
- Problem: Semantic Heterogeneity -> Semantic Web



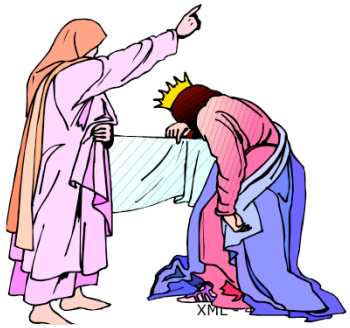
- Cost of semantic integration: 1 Trillion$ / year (Mike Brodie, GTE)

XML provides a framework to encode data on the Web in a standardized fashion, to deal with irregular Web data structures and to exchange Web data among information systems. Thus it contributes in particular to overcoming syntactic heterogeneity on the Web.

The markup used in XML can be interpreted by applications. This enables automatic processing of Web data. However, for properly interpreting the markup, semantic heterogeneity needs to be overcome as well. The same concepts can (and will be) represented in different application contexts differently, e.g., by using different terms to denote the same meaning. This is a major obstacle to enable interoperability on the Web and indeed in industry this is considered as one of the major problems in the use of information technology.

The Vision of W3C: Semantic Web

- The *Semantic Web* is an extension of the current Web in which *information is given well-defined meaning*, better enabling computers and people to work in cooperation. The mix of content on the Web has been shifting from exclusively human-oriented content to more and more data content. The Semantic Web brings to the Web the idea of having data defined and linked in a way that it can be used for more effective *discovery, automation, integration, and reuse across various applications*. For the Web to reach its full potential, it must evolve into a Semantic Web, providing a universally accessible platform that allows data to be shared and processed by automated tools as well as by people. The Semantic Web is an initiative of the World Wide Web Consortium (W3C), with the goal of extending the current Web to facilitate Web automation, universally accessible content, and the 'Web of Trust'. (http://www.w3.org/2001/sw/Activity)



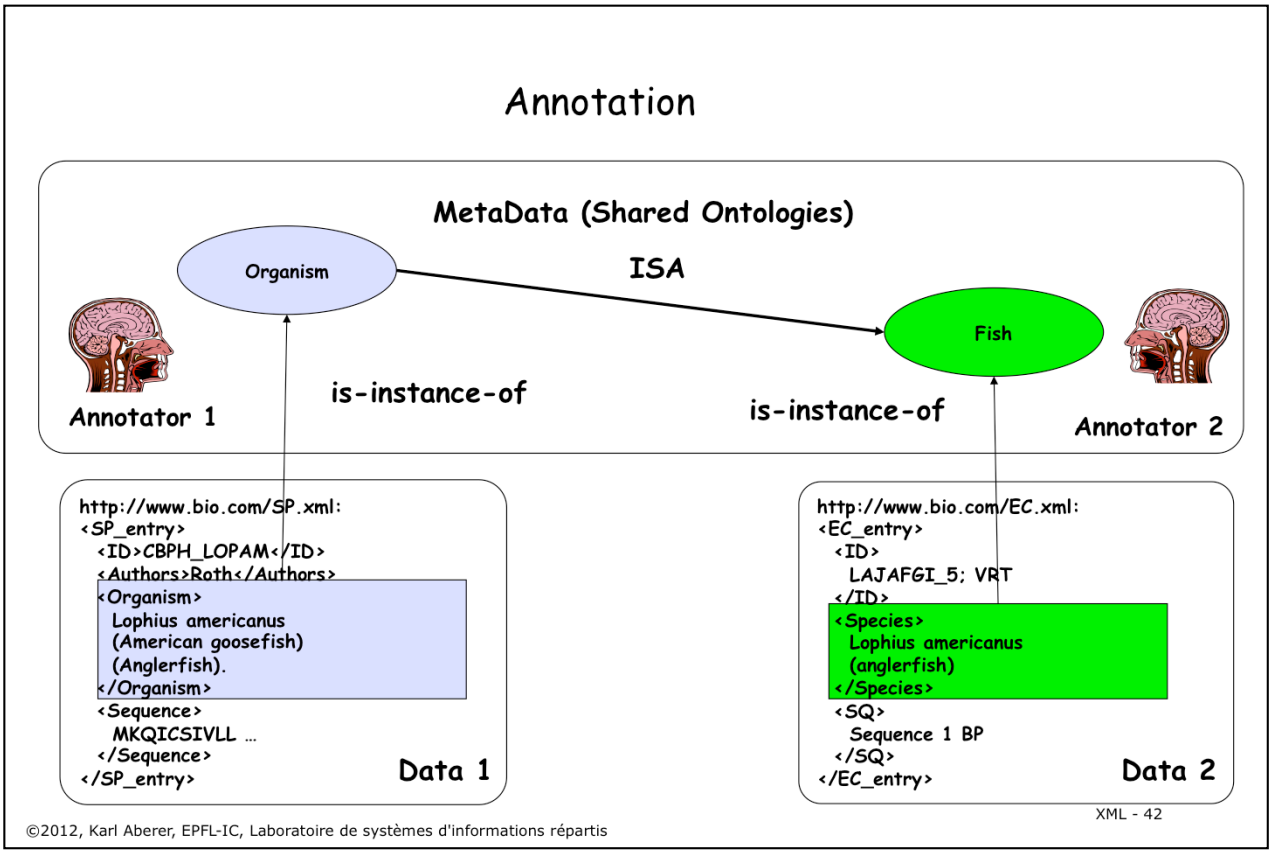
In order to address the problem of semantic interoperability and thus enable automated processing of Web data, the W3C initiated the "Semantic Web" initiative. The Semantic Web is a framework that builds on Web technology, including the XML framework and extends it with technologies that facilitate semantic interoperability.

Three Ways to Overcome Semantic Heterogeneity

1. Standardization: agree on common user-defined markup (schemas)
   - great if no pre-existing applications
   - great if power player enforces it
2. Translation: create mappings among different schemas and databases
   - requires human interpretation and reasoning
   - mappings can be difficult, expensive to establish
3. Annotation: create relationships to agreed upon conceptualizations
   - requires human interpretation and reasoning
   - annotation can be difficult, expensive to establish
   - reasoning over the conceptualization can provide added value

There exist three basic approaches to tackle the problem of automating semantic processing of the data on the Web:

1. Standardization avoids the problem of semantic heterogeneity a priori. This approach is used when there exists already (historically) a wide agreement on the structure of relevant data and their interpretation. For example, terminology in the financial insustry is largely standardized, and therefore it is not a major problem to come up with agreed upon formal specifications of such terminology and corresponding schemas to represent the data. This is even more the case as there exist in this type of business environment typically strong players that can enforce the standards.
2. Translation or mapping between different schemas and databases is the second possibility to establish semantic interoperability. This is the approach that has been extensively studied for data integration problems in relatively small and controlled domains, such as inside businesses and organizations. The requirements in these domains are typically changing not too quickly, thus much effort can be invested into developing the necessary mappings in order to properly map data from one representation into another, or to map data from multiple representations into one common global representation.
3. The third possibility is slightly different from the second: instead of engineering mappings between heterogeneous schemas for each integration problem, one first agrees on a common conceptualization of the domain, covering relevant aspects for a large class of applications. This conceptualization is normally called an ontology and is supposed to be application independent. Since ontologies have a formal representation they are machine-processable. Once such an ontology is in place existing information sources can relate their structural elements for expressing certain concepts (e.g. element names) to concepts from the ontology. This then (ideally) enables other applications to properly interpret the contents of the information system. In addition, ontologies in the general case should include reasoning capabilities, which would permit to use not only hard-coded, or pre-canned knowledge (e.g. in form of explicitly relating concepts from an information system to the ontology), but also to derive new knowledge from combining existing knowledge in different ways such that new implied relations can be derived that have not been explicitly provided.



This differentiation among different approaches to semantic interoperability can be nowadays found in standardization documents: according to ISO 14258 (Concepts and rules for enterprise models), there are three basic ways to relate entities together:

Integrated approach. There exists a common format for all models. This format must be as detailed as the models themselves. The common format is not necessarily a standard but must be agreed upon by all parties in order to elaborate models and build systems. (-> Standardization)

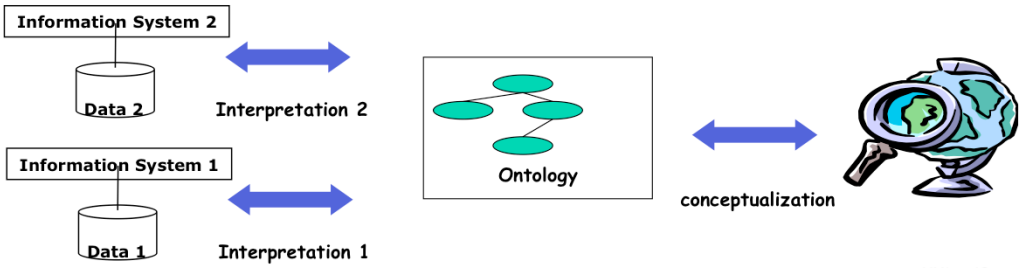
Federated approach. There is no common format. In order to establish interoperability parties must accommodate on the fly. Federated approach implies that no partner imposes their models, languages and methods of work. This means they must share a common ontology. (-> Translation)

Unified approach. There exists a common format but only at a meta-level. This meta-model is not an executable entity as in the case of the integrated approach but provides a way for semantic equivalence to allow mapping between models. (-> Annotation)

This simple example illustrates of how ontologies might help to increase semantic interoperability and how new knowledge may be generated by reasoning. Take our earlier example of biological databases. These typically use different schemas to model related facts. For example, Database 1 uses the term Organism to denote an organism, and database 2 uses the term Species to do the same. Two annotators, who share the same ontology, now inspect the document and each of them associates the elements with terms taken from the ontology. So Annotator 1 decides that the element Organism corresponds to information related to an organism, whereas annotator 2 recognizes actually from the content that the element is related to information about a fish and annotates correspondingly (by establishing a is-instance-of relationship). At this point, the fact that both annotators used the same ontology and that reasoning is possible in this ontology comes into play. Since in the ontology a Fish is a subconcept of Organism (a fact represented formally by a ISA relationship in the ontology) an automated processing tool (e.g. for searching for information) might exploit this relationship and correctly identify for a request for information on Organisms in both databases the related elements.

Ontologies

- Ontologies are an explicit specification of a conceptualization of the real world (Gruber, 1993)
- Ideally
  - different information systems agree on the same ontology
  - relate their model/schema/data elements to the ontology
  - mapping can be constructed via the ontology
- Issues
  - ontology languages (e.g. RDF), ontology engineering
- Problem: requires agreement on the conceptualization of the real world !



Thus ontologies provide a “proxy representation” for the real world , to which the interpretation of data in information systems refers to and thus provide an interpretation of the data in an information system that can be automatically processed. In order to realize this vision a number of technical challenges need to be addressed.

Issues for Ontologies

- Agreement on meaning of terms
  - concepts: what does organism mean ? what does fish mean ?
- Modeling and encoding of ontologies
  - modeling primitives: what does an arrow mean ?
    what does "instance-of" mean ? what does ISA mean ?
  - different encodings of the model

```
<is-instanceof uri="http://www.bio.com/SP.xml/organism">
  Organism
</is-instanceof>
```

```
<annotate  uri="http://www.bio.com/SP.xml/organism>
          <with label="is-instanceof"/>
          <with value="Organism"/>
</annotate>
```

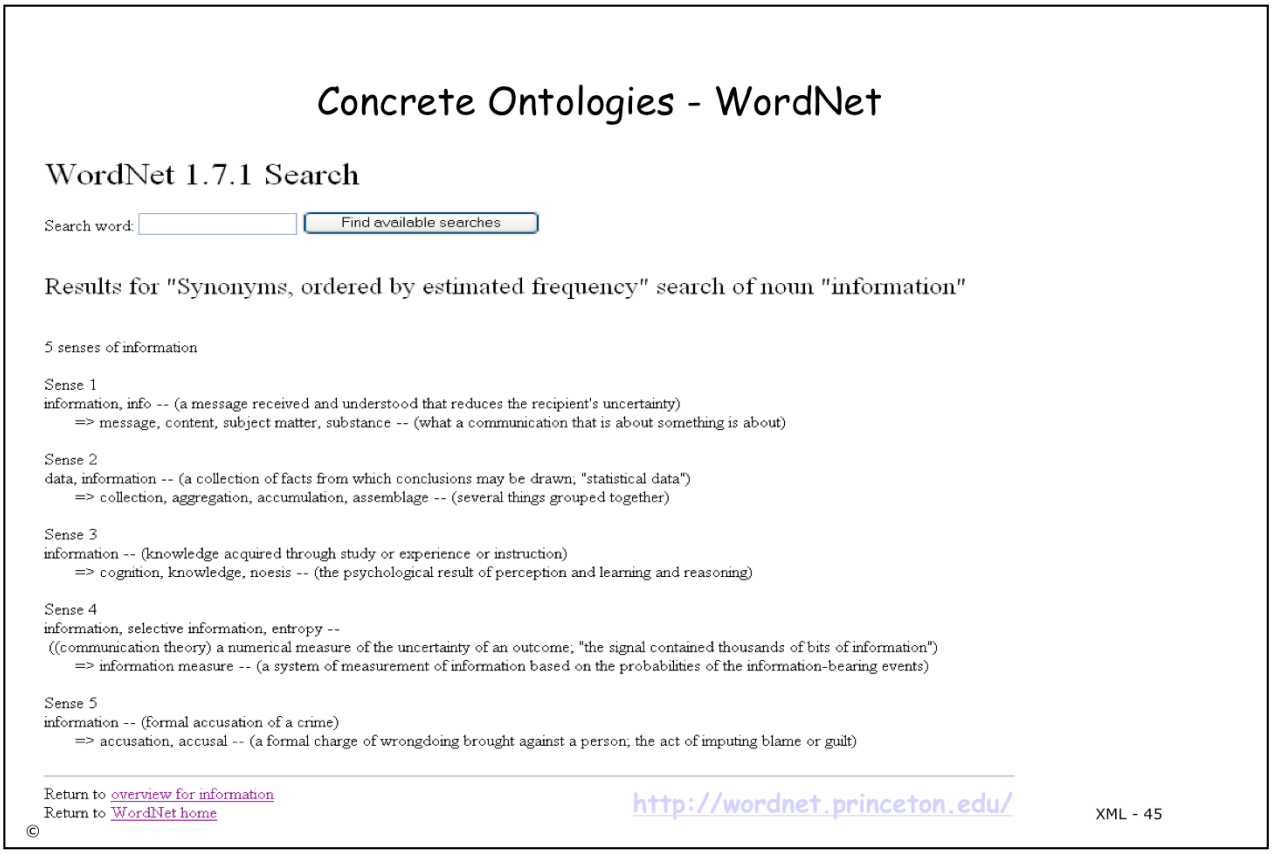
```
<Organism anotate="is-instanceof"
  value=" http://www.bio.com/SP.xml/organism ">
    Lophius americanus
    (American goosefish)
    (Anglerfish).
</Organism>
```

```
<is-instanceof
    uri="http://www.bio.com/SP.xml/organism"
    value="Organism"/>
```


What are the important issues for enabling the use of ontologies?

1. Ontologies need to be built: this requires an agreement on the meaning of terms (symbols). In practice, there exists no other way to establish such an agreement than by extensively collecting knowledge (from humans) and represent it within a formal specification. This has been done in practice and as a result there exist extensive ontologies, some of them have been built up over many years (see the example on the next slide)
2. Ontologies need to be represented in a language or data model: in order to store the ontology a representation mechanism (a model) is needed, and the model needs to be encoded into data. The choice of the model is an important issue, since it should be very expressive (as a wide variety of aspects of the real world need to be modeled) and easy to use (as ontologies should be used in a wide range of applications) at the same time. The encoding is equally important, as it should be done in a standardized form. If this where not the case we would immediately loose the advantage of having a common conceptualization of the world at the abstract level, since we were not able to exchange it properly with others. The examples illustrate of how using different encodings of the same fact in an ontology can lead to widely varying representations, which in turn may lead to problems of proper interpretation and processing.

Concrete Ontologies - WordNet



WordNet 1.7.1 Search

Search word: Find available searches

Results for "Synonyms, ordered by estimated frequency" search of noun "information"

5 senses of information

Sense 1
information, info -- (a message received and understood that reduces the recipient's uncertainty)
=> message, content, subject matter, substance -- (what a communication that is about something is about)

Sense 2
data, information -- (a collection of facts from which conclusions may be drawn; "statistical data")
=> collection, aggregation, accumulation, assemblage -- (several things grouped together)

Sense 3
information -- (knowledge acquired through study or experience or instruction)
=> cognition, knowledge, noesis -- (the psychological result of perception and learning and reasoning)

Sense 4
information, selective information, entropy --
((communication theory) a numerical measure of the uncertainty of an outcome; "the signal contained thousands of bits of information")
=> information measure -- (a system of measurement of information based on the probabilities of the information-bearing events)

Sense 5
information -- (formal accusation of a crime)
=> accusation, accusal -- (a formal charge of wrongdoing brought against a person; the act of imputing blame or guilt)

Return to overview for information
Return to WordNet home

http://wordnet.princeton.edu/

XML - 45

Originally ontologies have been developed in the field of artificial intelligence, e.g., to facilitate knowledge sharing, reasoning and reuse and natural language processing. More recently, ontologies are becoming more widely used in fields such as intelligent information integration, cooperative information systems, information retrieval, and electronic commerce.

One example of an ontology that has been developed as part of these efforts is WordNet. WordNet explains the meaning of English terms in a way as general and as precise as possible. The example gives the different meanings of the term "information" in English. WordNet can be freely accessed and downloaded over the Internet. It has become very popular for many purposes, including data integration and increasing the quality of Web search engines.

Model Requirements for Ontologies

	HTML	XML	RDF	OWL
Simplicity	+	+	+	+
Exchangeability	+	+	+	+
Non-intrusive annotation	+	-	+	+
Domain-specific vocabularies	-	+	+	+
Modeling primitives	-	-	+	+
Reasoning capabilities	-	-	-	+

"separate structure from presentation"

"separate interpretation from structure"

XML - 46

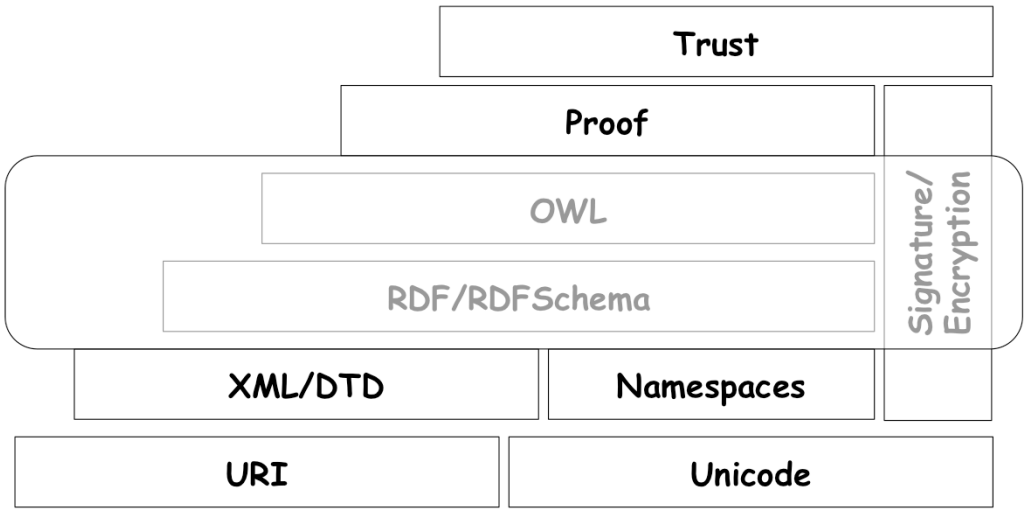
With respect to modeling and encoding of ontologies for the Semantic Web, there exist a number of requirements, some of which follow from what we have discussed earlier:

1. Simplicity: the success of the Web was always founded on the principle of simplicity of concepts to encourage wide-spread use. Therefore complex models will not be successful. This is an important criterion, since some of the existing ontologies (one example is Cyc) are expressed in fairly complex knowledge representation models.
2. Exchangeability: Since the web is a communication environment, any kind of data that is processed must be easily exchangeable. This is what motivated the use of XML as a data representation format in the first place and should hold for metadata and ontology data as well.
3. Non-intrusive annotation: as the example on annotation we gave earlier demonstrated, machine-processable knowledge required for the interpretation of data will be associated with the data typically a-posteriori. Also there does not always exists a unique interpretation for the same data. Therefore any attempt to encode the knowledge required for interpretation directly into the data is not practical. This excludes, for example, the approach to use XML elements for annotation.
4. Domain-specific vocabularies: the model must provide a mechanism that permits to introduce vocabulary or terminology that is specific to a domain, in other words the possibility to specify schemas for different domains.
5. Modeling primitives: since an ontology model will be used in many different, and potentially very complex contexts (applications) they have to offer a sufficiently rich set of possibilities to model complex situations (e.g. complex structures or complex relationships). There exists a rich experience in modeling (e.g. from data modeling in databases, e.g. the entity-relationship model) and models for ontologies can draw from them.
6. Reasoning Capabilities: the example we discussed earlier already illustrates that even simple forms of reasoning within the ontology layer can make the interpretation of the data much more powerful (and thus the processing in the Semantic Web).

In this table we evaluate HTML, XML, RDF and OWL with respect to each of these aspects. RDF is the Resource Description Framework and is the first WWW standard proposed for the Semantic Web. OWL is the ontology interchange language, an extension of RDF proposed to enrich it with more reasoning capabilities and providing a well-defined semantics. We will introduce both models subsequently.

An interesting interpretation of the role of ontologies in the Web architecture is the following: ontology models support automated processing of semantics on the Web, and thus separate semantic concerns from the concern of structuring data; This is similar to the step from HTML to XML, were the issues of structuring data where separated from the issues related to the presentation (layout) of data. With the Semantic Web thus an attempt is made to separate meaning from structure.

W3C Web architecture



XML - 47

The Semantic Web standards RDF and OWL are positioned in the Semantic Web architecture in top of the syntactic layer.

Question

- An ontology
  1. Helps to separate layout issues from the structural representation of data
  2. Provides a common syntactic framework to represent standardized domain models
  3. Can be used as an annotation framework for semantically hetergeneous databases

4. RDF Resource Description Framework

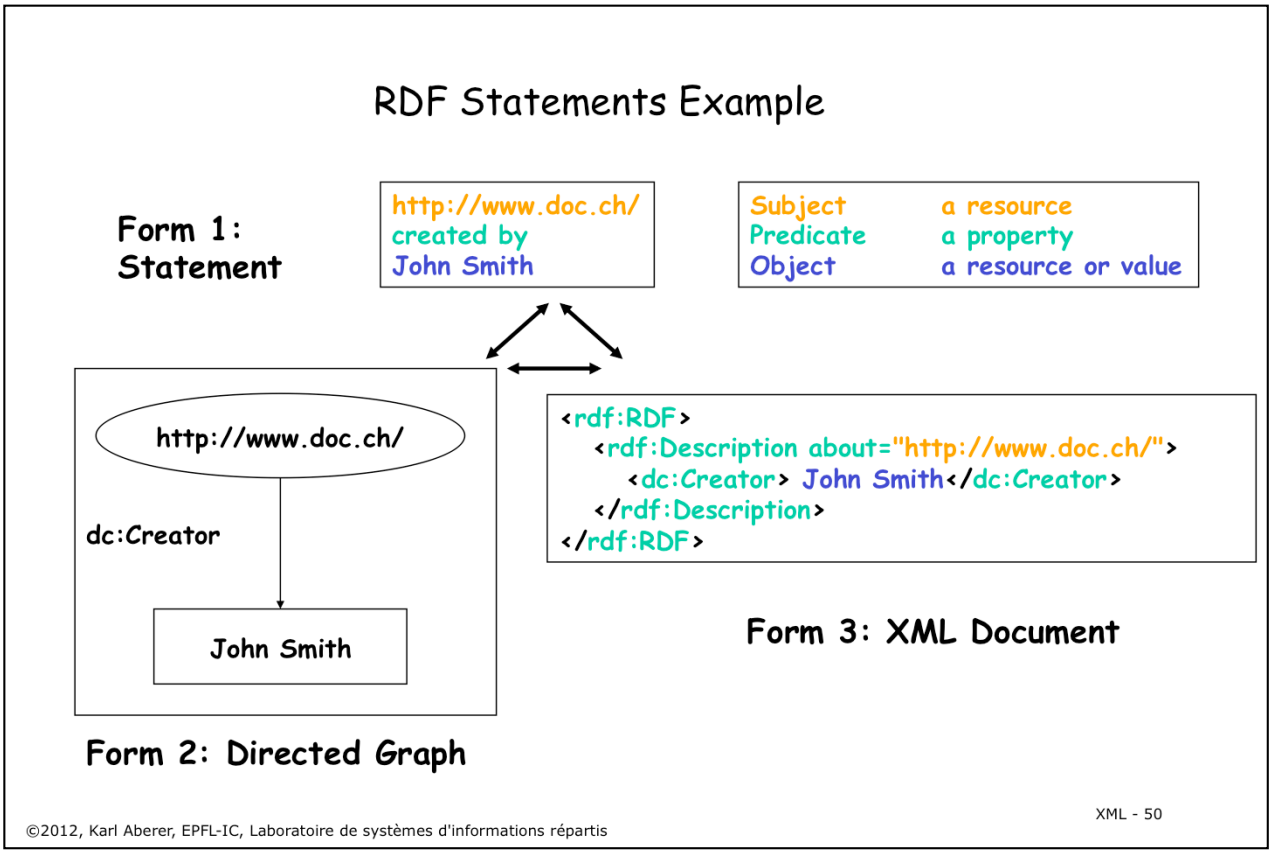
- RDF (Instances)
  - *Statements* about *Resources* (addressable by an URI) and literals (XML data)
  - Statements are of the form: subject property object
  - Like simple natural language sentences
  - RDF statements are themselves resources (reasoning about RDF)
  - *Properties* define relationships to other resources or atomic values
- RDF-Schema
  - Data model to specify schemas for RDF instances
  - Which properties are applicable for which objects with which subjects
  - Defines "grammar" and "vocabulary" for semantic domains (ontology language)
- Relation RDF vs. RDFS comparable to well-formed vs. valid XML

RDF is a standard supported by the W3C (http://www.w3.org/RDF/) to represent metadata. It is a fairly simple, graph-oriented data model to annotate any kind of XML document.

RDF consists of two parts: a language for representing metadata instances (RDF), which allows to annotate Web resources with statements. The Web resources are addressed by Universal Resource Identifiers (URI), of which URL's are the most important example. Thus any Web document or part of it can be annotated. The second part of the RDF standard is a language for specifying schemas for RDF Instances. This language enables specification of the vocabulary and grammar that is used for forming statements for annotation. Since RDF statements can be created also without using RDF schemas, RDF is a semi-structured data model, similar as with well-formed XML (instances) and XML-DTD (schemas).

The RDF model is similar to the entity-relationship (ER) model. Entities correspond to resources and relationships correspond to properties. The main difference is that RDF requires that relationships are directed, and have a specific semantics: the resource from which the (directed) relationship emerges is assigned a property with the value to which the relationship points. This reflects the intention to use RDF to associated metadata (the value) with data (the source of the relationship). Sample RDF applications include PICS (annotating documents with information on the suitability of the content for certain groups, e.g. like the movie rating system) and Dublin Core (annotating documents with basic bibliographic information.

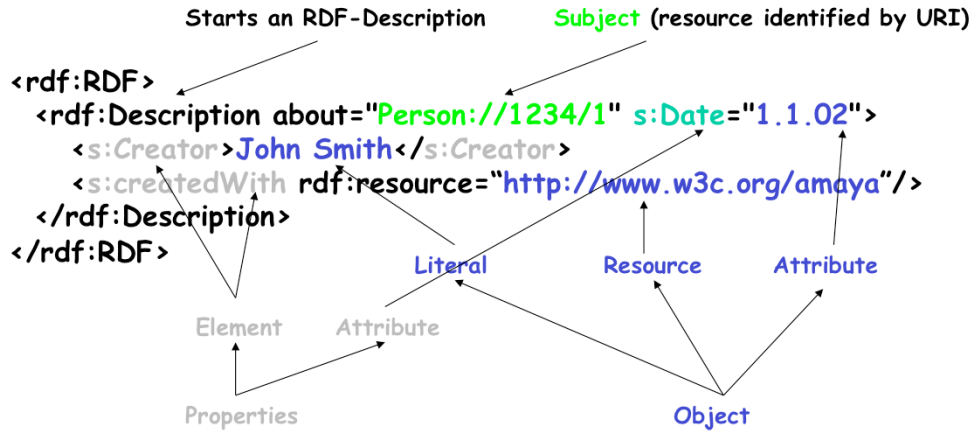
It is important to know that the syntax of RDF and its encoding into XML is well-specified, but that the semantics RDF is only specified in a « semi-formal » manner. This introduces certain ambiguities for applications interpreting RDF statements.



The basic constituent of RDF is an RDF statement. We can view RDF statements in three different ways: we can view them like natural language sentences, where the subject is a URI (uniform resource identifier) and the object can be either a URI or a String; we can view them as directed graphs, where the subject and object are represented as nodes (an ellipsis is used to represent resources (identified by a URI) and rectangles to represent literals (atomic XML values) and the predicate is represented as directed link , or we can view them as XML documents where the RDF statement is encoded into XML format. The graph representation is in particular suitable for visual presentation and reasoning, whereas the XML representation is used for exchange and storage.

RDF Syntax

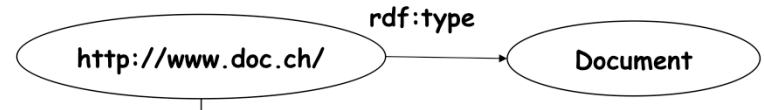
- Many syntactic varieties possible
- Basic form



For encoding RDF into XML there exist many syntactic variations, which can make the understanding of RDF documents sometimes rather difficult. Here we summarize the most important variants. The basic pattern of encoding is as follows: the subject is represented by an XML element called rdf:Description. This element is the root of the document fragment representing the RDF statement. In the content of this element one finds one (or more) predicates, represented by XML elements, e.g. s:Creator. The content of this XML element in turn represents the object of the RDF statement. If the object is not a literal, one can alternatively represent the object as an attribute of the predicate element. Also, both the predicate and the object can be encoded into the element representing the statement, as it is shown for s:Date predicate.

Typing Resources

- Resources can be associated with a type by using **rdf:type** (a special property)



dc:Creator

John Smith

```
<rdf:RDF>
  <rdf:Description about="http://www.doc.ch/">
    <rdf:type>Document</rdf:type>
    <dc:Creator> John Smith </dc:Creator>
  </rdf:Description>
</rdf:RDF>
```

abbreviated syntax

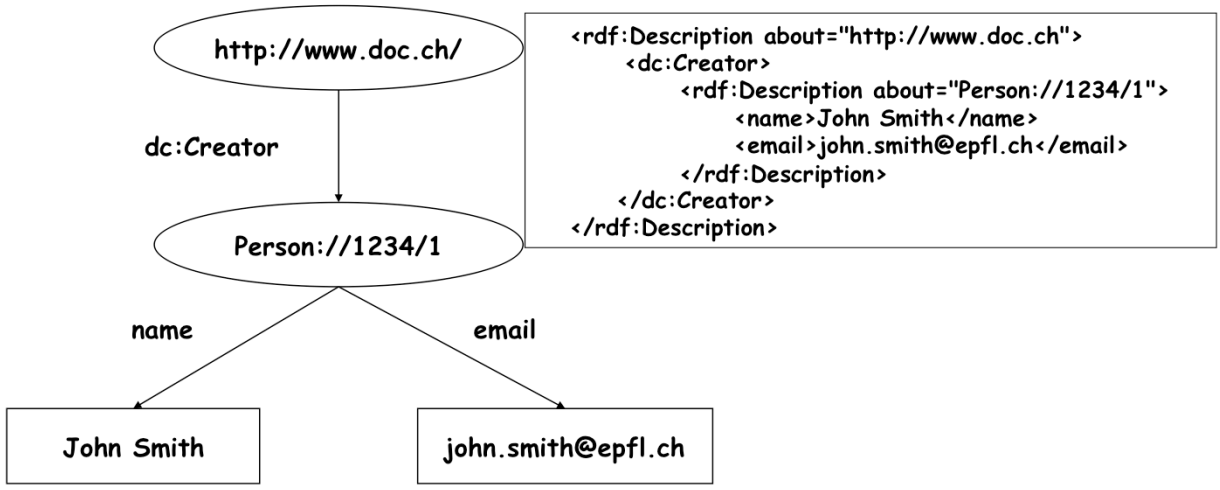
```
<rdf:RDF>
  <dc:Document about="http://www.doc.ch/">
      <dc:Creator> John Smith </dc:Creator>
  </dc:Document>
</rdf:RDF>
```

XML - 52

RDF allows us to categorize resources into different classes (typing). For that purpose one associates with the resource that should be categorized another resource, that represents the category, using a special RDF property rdf:type. We will see later, when introducing RDF schema, what are possible uses of typing. With respect to encoding, the type property can either be represented explicitly like any other property, or one can use a special abbreviated syntax, where the name of the type becomes the element name of the element representing the statement.

RDF Complex Values

- Use an intermediate resource

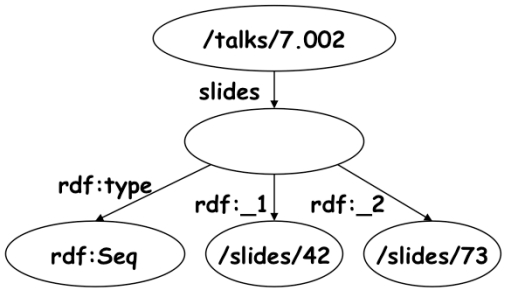


Associating a subject with a property whose value is a simple object is just the simplest form of a statement. In general, the value of a property (the object of the statement) may be a complex statement itself (in data type parlance “of a complex type”). In order to represent such complex object values a new intermediate resource is created (in the example Person://1234/1) to which different properties are associated. Note that this is different of directly associating these properties with the subject of the overall statement (http:// www.doc.ch/ in the example) (why?).

In the XML encoding such a complex object value can be represented by directly inlining the complex object into the content of the statement.

RDF Containers

- Containers
  - Bag (unordered)
  - Seq (ordered)
  - Alt (alternatives)
- Quantifiers
  - about: John is author of the talk (consisting of many slides)
  - aboutEach: John is author of each slide of the talk



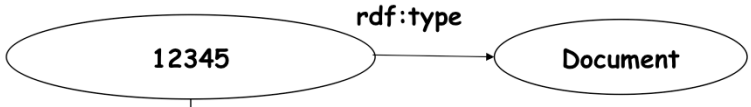

```
<rdf:RDF>
  <rdf:Description about="/talks/7.002">
    <s:slides>
      <rdf:Seq>
        <rdf:_1 resource="/slides/42"/>
        <rdf:_2 resource="/slides/73"/>
      </rdf:Seq>
    </s:slides>
  </rdf:Description>
</rdf:RDF>
```

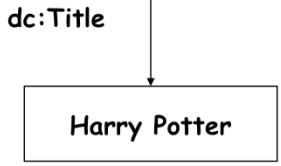

Statements can be made not only using single resources, but as well using collections of resources. For that purpose RDF provides the container concept. Containers are special resources of one of three container types that are specified in the RDF standard. A container resource is then associated with a set of other resources. By creating a statement using the container object as "object", one can express statements made about the set of objects. More precisely, one can specify whether the statement is a statement of the set of objects "as a whole" or a statement that applies to each element of the set individually (what the consequences of this distinction are is not further specified in RDF, this is an example where the semantics of RDF is not clearly specified).

There exist three different types of containers: bags which are unordered multi-sets (= sets with multiple occurrences of the same resources), sequences which are ordered sets (i.e. lists) of resources and alternatives which is a single resource that is to be chosen out of a given set. The property labels can be used to impose an order on the elements of the set, by using labels _1, _2 etc. If the order is irrelevant one can use the alternative syntax rdf:li instead of rdf:_1, rdf:_2 etc.

Creating New Resources

- New RDF resources are created by using **rdf:ID** (a special property)






```
<rdf:RDF>
  <rdf:Description rdf:about="#12345">
    <rdf:type>Document</rdf:type>
    <dc:Title>Harry Potter</dc:Title>
  </rdf:Description>
</rdf:RDF>
```

Alternative syntax

```
<rdf:RDF>
  <rdf:Description rdf:ID="12345">
    <rdf:type>Document</rdf:type>
    <dc:Title>Harry Potter</dc:Title>
  </rdf:Description>
</rdf:RDF>
```

```
<rdf:RDF>
  <dc:Document rdf:ID="12345">
     <dc:Title>Harry Potter</dc:Title>
  </dc:Document>
</rdf:RDF>
```

abbreviated syntax

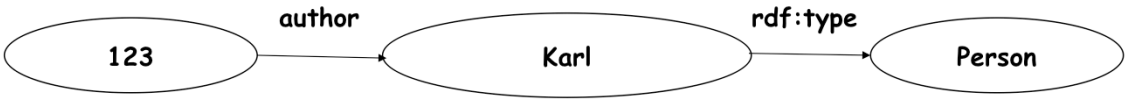
RDF resources need not necessarily be pre-existing Web resources identified by URIs, but can also be instantiated within RDF statements, i.e., *new resources* can be defined as part of RDF statements. A resource is in general anything that can be *identified* on the Web. As a consequence also a new resource requires foremost an identifier. For this purpose RDF provides the rdf:ID attribute to introduce the identifier of the resource. This attribute replaces rdf:about attribute when the statement is about a new resource instead about an existing resource. Using the identifier, this new resource can then be referred to in other RDF statements.

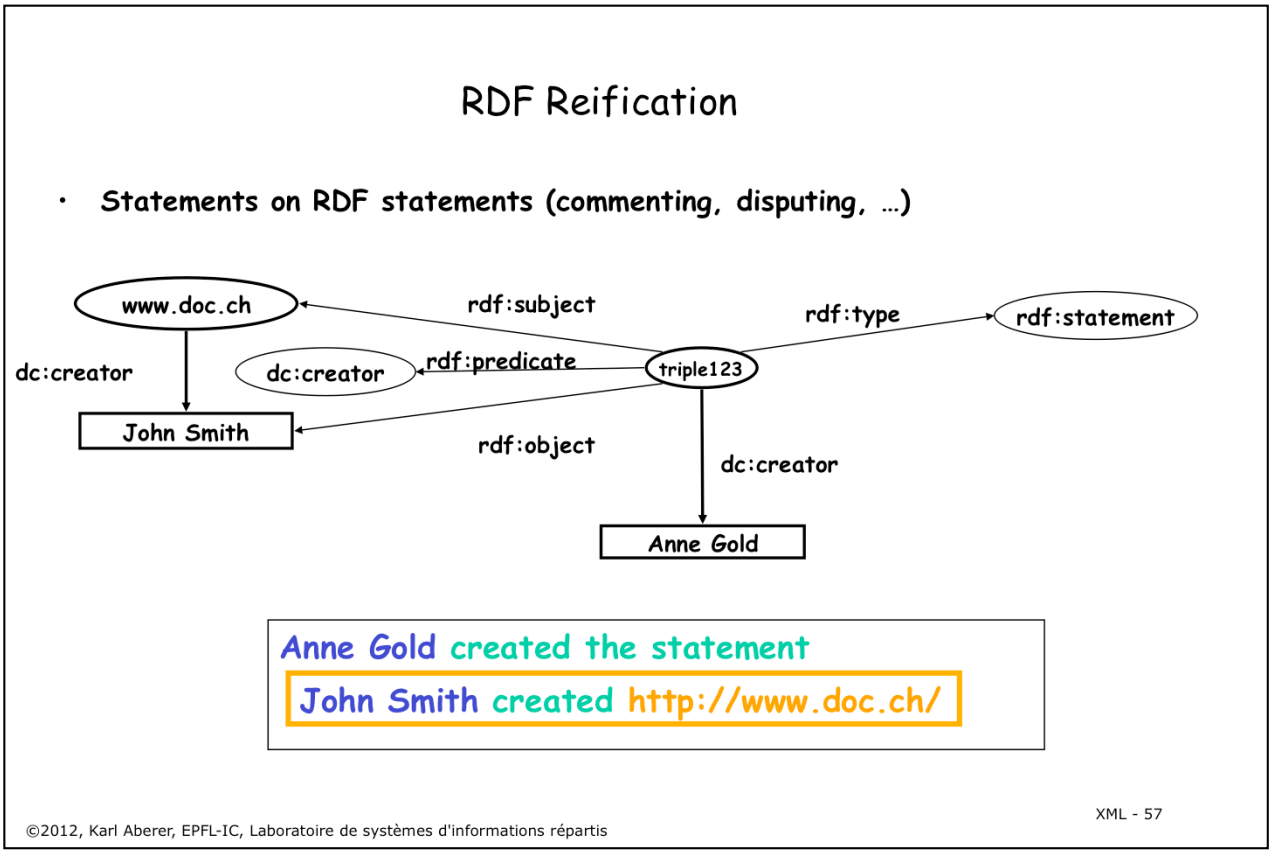
Question

- Which abbreviated syntax of the following expression is wrong?

```
<rdf:RDF><rdf:Description rdf:ID="123"><author>
        <rdf:Description rdf:ID="Karl"><rdf:type>Person</rdf:type>
    </rdf:Description></author></rdf:Description></rdf:RDF>
```

1.
```
<rdf:RDF><rdf:Description rdf:about="#123"><author>
    <rdf:Description rdf:about="#Karl"><rdf:type>Person</rdf:type>
    </rdf:Description></author></rdf:Description></rdf:RDF>
```
2.
```
<rdf:RDF><rdf:author rdf:ID="123">
  <rdf:Description rdf:ID="Karl"><rdf:type>Person</rdf:type>
    </rdf:Description></rdf:author></rdf:RDF>
```
3.
```
<rdf:RDF><rdf:Description rdf:ID="123"><author>
    <rdf:Person rdf:ID="Karl"></rdf:Person></author></rdf:Description></rdf:RDF>
```





In RDF everything is a resource. In particular, RDF statements are considered as resources and therefore it should be possible to make statements about them. From the viewpoint of the Semantic Web this is in fact extremely important. Since annotations do not express absolute truth but rather different interpretations of data, it is important to foresee the possibility of annotating annotations (such as illustrated in the simple example). This allows to comment on annotations, to agree or dispute them etc..

When considering the graph representation of RDF, it is not immediately clear of how to treat a statement as a resource, since a statement consists of three structural elements, the subject, the object, and the predicate. But we can apply the same "trick" as we did already for complex objects and collections. We introduce a new resource which serves as representative for the statement. This resource obtains as properties all the constituents that make up the statement it represents. This process is called *reification*. It is straightforward to connect the resource representing the statement it to it's subject and object, as they are both resources themselves. Also the type of the object can be determined through a property rdf:type pointing to the special resource rdf:statement representing the category of statements. For the predicate a specific new resource representing the predicate is required. We will see later, when introducing RDF schema, that this new resource is indeed part of a schema for RDF statements, expressing that statements using this properties are possible in the given context. The reified RDF statement is connected to its constituents through the special RDF properties rdf:object, rdf:subject and rdf:property. By reifying a statement one creates a new resource, which can be anonymous, if no identifier is associated with it using rdf:ID, or can be referenced if it has such an identifier.

RDF Reification - Syntax

- The statement has an anonymous resource as subject, namely the reified statement which is fully characterized by its properties !

```
<rdf:RDF>
  <rdf:Description about="#triple123">
    <rdf:subject resource="http://www.doc.ch"/>
    <rdf:predicate resource="http://description.org/schema#Creator"/>
    <rdf:object>John Smith</rdf:object>
    <rdf:type resource="http://www.w3.org/TR/WD-rdf-syntax#Statement"/>
    <dc:Creator>Anne Gold</dc:Creator>
  </rdf:Description>
</rdf:RDF>
```

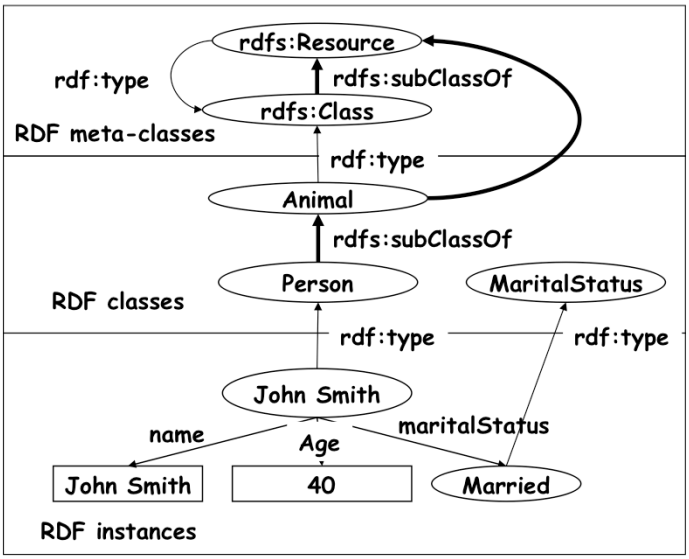

The XML encoding of reified statements follows the principles that have been introduced before. The reified statement is represented as a complex, anonymous object.

Question

- Which is true?
  1. Reification is used to produce a more compact representation of complex RDF statements
  2. Reification is needed to make a statement the subject of another statement
  3. Reified statements always make a statement about another statement

RDF Schema - Classification

- RDF resources
  - anything that can be described
- RDF classes
  - Categories for subjects and objects
  - Classes allow to associate a type with an RDF instance
  - Different classes can be in a subClass relationship (be included in each other)
  - Classes are themselves RDF instances



We now introduce RDF Schema. RDF Schema provides two basic mechanisms.

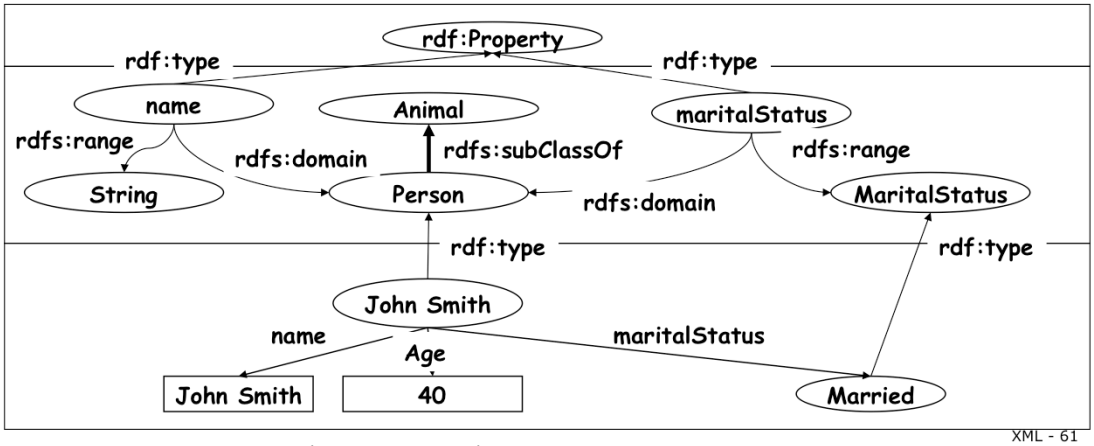
1. Categorization RDF resources, into *classes*.
2. Constraints on the possible use of properties, in the form of constraints expressing which classes can participate as subjects and objects in statements using a specific property.

First we describe the classification mechanism. Classes are represented themselves as resources, which are of type rdfs:Class, a special class in the RDFS specification. The type property rdf:type is used (as in RDF) to indicate the type of a *class resource*. If an application resource is of a specific type then the resource is connected to the corresponding class resource via the rdf:type property. In the illustration two examples of this are included: the resource with ID „John Smith" belongs to the class (or is of type) Person and the resource "Married" is of type MaritalStatus (which is a resource representing a specific predicate type). Between different classes a subclass relationship can be specified, by using the attribute rdfs:subClassOf. The intended semantics is that any resource belonging to the subclass also belongs to the superclass (containment relationship).

An interesting aspect of RDFS is the modeling of the RDF and RDFS concepts within RDFS itself. This is done by introducing a meta-class level that models the modeling constructs and reflects the paradigm that in RDF everything is a resource, including the concepts introduced by RDF and RDFS. In particular classes are sets of resources, and thus the rdfs:Class resource is a subClass of rdfs:Resource. Application classes, such as "Animal", are sets of resources, and thus also subclass of rdfs:Resource. On the other hand, the type of a resource is indicated by the rdf:type property. Since rdfs:Resource is a class it is connected via the rdf:Type property to the rdfs:Class resource. This produces the cyclic structure that we can observe within the RDF meta-class schema (in fact in the figure only a small fragment of the RDFS meta-class schema is shown).

RDF Schema - Properties

- RDF properties
  - Connect resources
  - The RDF instance must have the properties that are declared for the class
  - rdfs:domain: classes of which the instances may have a property
  - rdfs:range: classes of which the instances may be the value of a property



XML - 61

The second important concept of RDFS is the possibility to constrain the usage of RDF properties that connect resources. As everything in RDF, RDF properties are resources themselves. Thus, they are represented in an RDF schema as resources. For example maritalStatus is a resource representing a property. In RDF schema it is now possible to constrain the usage of properties as follows: by connecting the property resource through the property rdfs:domain to a class resource, one specifies that the subject when using this property must originate from that class, i.e., be of the type of this class. Similarly for the object, the range can be constrained using rdfs:range. Ranges can also be of atomic type, in that case one connects the property resource to (predefined) resources representing the data type of the atomic type. In the example above this is the atomic type STRING.

The RDFS model bears a lot of similarity with object-oriented models (or type specifications in the context of OO programming languages). However a fundamental difference is that properties (attributes in OO terminology) are defined independently of classes.

RDF Schema - Syntax

```
<rdf:RDF xml:lang="en"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"        xmlns:rdfs="http://
www.w3.org/2000/01/rdf-schema#">
<rdfs:Class rdf:ID="Person">
              <rdfs:subClassOf rdf:resource="http://www.w3.org/classes#Animal"/>
</rdfs:Class>
<rdf:Property ID="maritalStatus">
        <rdfs:range rdf:resource="#MaritalStatus"/>
        <rdfs:domain rdf:resource="#Person"/>
</rdf:Property>
<rdf:Property ID="name">
        <rdfs:range rdf:resource="http://www.w3.org/classes#String"/>
        <rdfs:domain rdf:resource="#Person"/>
</rdf:Property>
<rdf:Property ID="age">
        <rdfs:range rdf:resource="http://www.w3.org/classes#Integer"/>
        <rdfs:domain rdf:resource="#Person"/>
</rdf:Property>
<rdfs:Class rdf:ID="MaritalStatus"/>
<MaritalStatus rdf:ID="Married"/>
<MaritalStatus rdf:ID="Divorced"/>
<MaritalStatus rdf:ID="Single"/>
<MaritalStatus rdf:ID="Widowed"/>
</rdf:RDF>
```


Since RDF schemas are expressed as RDF statements they can be encoded into XML along the same principles that we have introduced for RDF statements earlier. This example shows the complete encoding of the RDF schema we have used in our example before. Throughout the schema the abbreviated syntax for statements is used, replacing the element name "Description" by the corresponding class name of the subject of the description. Note of how using the ID attribute, in the RDF schema new resources are introduced. They can be referred to from other statements using the newly introduced identifier prefixed with #. The specification of the class with ID "MaritalStatus" includes the specification of the complete extension of the class, enumerating all possible values the members of this class can take, i.e., all possible predicate names that predicates of type MaritalStatus can take. Strictly speaking, the statements creating the instances of class are not part of the schema level but part of the instance level of RDF.

RDF Schema Inheritance

- rdfs:subClassOf
  - *A* subClassOf *B*: Every instance of *A* is also instance of *B*
  - transitive, not reflexive, anti-symmetric (no cycles!),
  - M:N: a class can have arbitrarily many subclasses and superclasses
  - Subclass has all properties of the superclass
- rdfs:subPropertyOf
  - *P1* subPropertyOf *P2*: If *A* has Property *P1* with value *B* then it has also value *B* with Property *P2*
  - Example: *Anne* has Property *Father* with value *John*, and *Father* subProperty *AParent* implies *Anne* has Property *AParent* with value *John*

Similar to object-oriented modelling, RDF provides inheritance mechanisms in order to support the reuse of specifications. RDF Schema provides both for type inheritance and property value inheritance.

1. **Type inheritance** is indicated by the rdfs:subClass property. If a class A is subclass of a class B this implies two things: first, every instance of class A is also an instance of class B, and second, every property that is specified for class A can also be used for class B (or is also specified for class B). Since the subclass relationship has the subset semantics it is transitive, but must not contain cycles.

2. **Property inheritance** is indicated by connecting two properties by the property rdfs:subPropertyOf. If a property P1 is subproperty of another property P2, this has the following meaning: every resource A that has property P1 (i.e. is the subject of a statement involving that property), and this property has value B (i.e. B is the object of the statement), then it has also property P2, an property P2 has the value B.

Question

- Which of the following are part of the RDF schema language?
  1. The « type » statement for RDF resources?
  2. The « domain » statement for RDF properties?
  3. The « subject » statement for RDF statements?

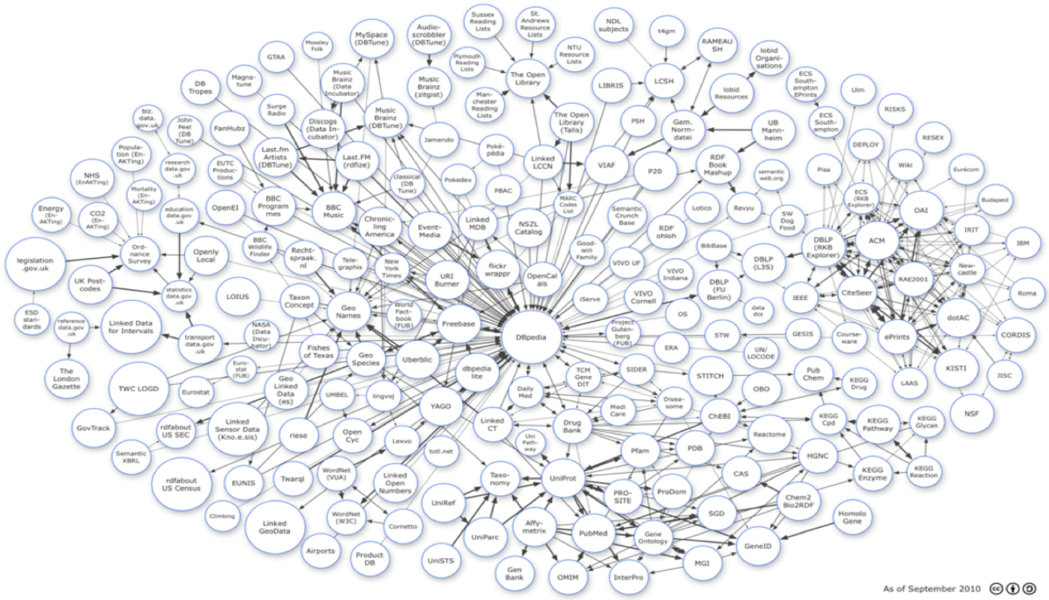
- Which is wrong? If a class is a subclass of another class
  1. Every element of the subclass is also an element of the other class
  2. Every property of the subclass is also a property of the other class
  3. Every subclass of the subclass is also a subclass of the other class

Comparison XML - RDF

Aspect	XML	RDF
Document/Instance	Nested String, Sequence	Graph, Sequence
Main application	Syntax Document content	"Semantics" Meta Data
Schema	Complex content models (Alternative, Sequence, Repetition)	(Complex) Objecttypes Resource = Object Property = Relationship
Inheritance	Not existent	SubClass SubProperty
Main application of schemas	Document Type Conformance	Vocabulary Specification

Both XML and RDF are two standards that define a data model. The both data models aim at different objectives, which results also in different properties of the models, which are listed in this comparison table.

Linked Open Data (linkeddata.org)



In the meanwhile begins to be widely adopted in order to publish datasets of general interests. In particular recent initiatives of the UK and US government to publish their data in RDF standards have heavily promoted this development. The linked open data initiative keeps track of this development and provides a nice overview of the main RDF data sources that are currently available on the Web.

5. Ontology Languages

- Limitations of RDF Schema
  - Limited expressive power (subclass, property, subproperty)
  - Unclear semantics (e.g. subproperty never has been precisely defined)
  - No reasoning support
- Requirements on ontology languages
  - Well **designed**
    - Intuitive to human users
    - Adequate expressive power
  - Well **defined**
    - Clearly specified **syntax** (obviously)
    - Formal **semantics** (equally important)
    - Adequate expressive power
  - **Compatible** with existing (web) standards
    - in particular RDF

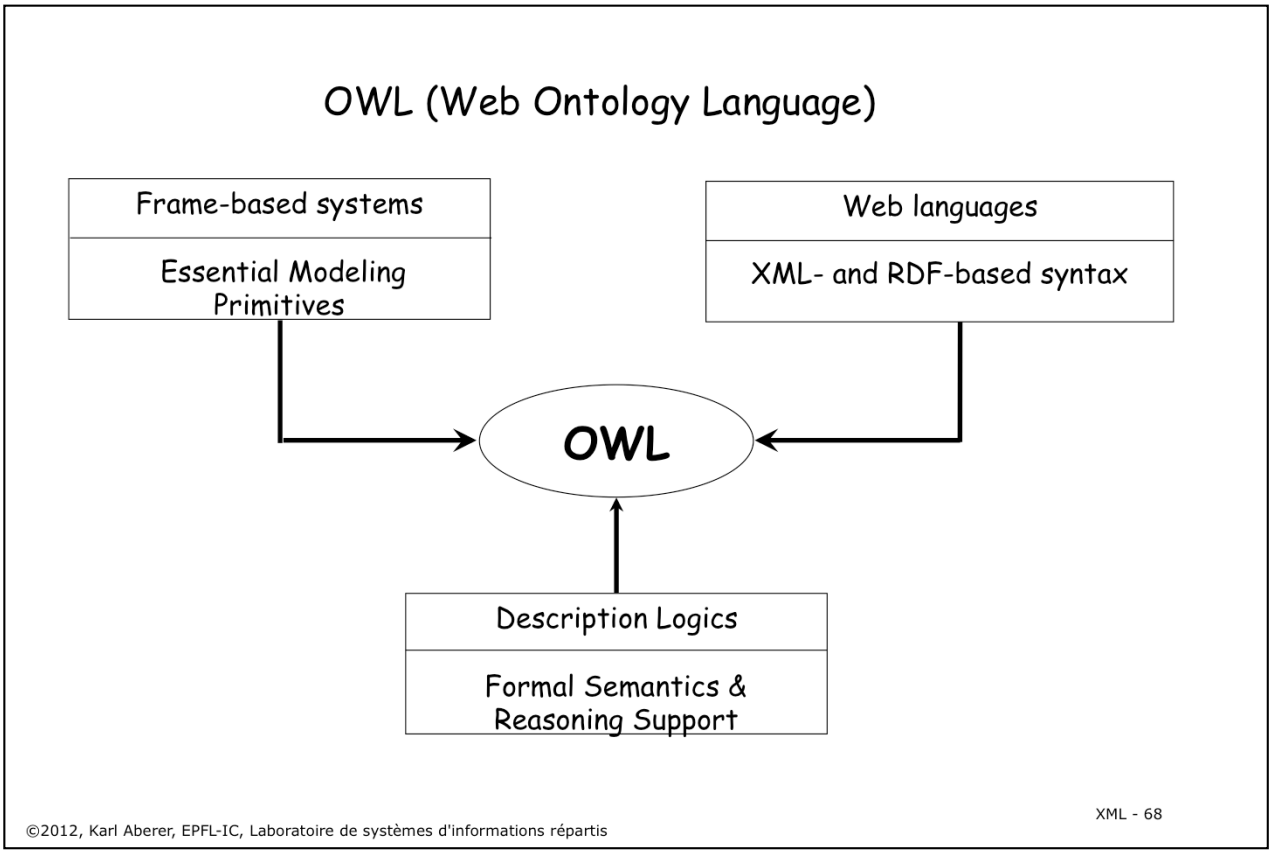
The main purpose of RDF is to serve as a conceptual data model for making the application semantics of a given domain explicit, in other words it is intended to serve as a language to express ontologies. For that purpose, however, it has been recognized that RDF has certain deficiencies: the set of basic modeling primitives is not sufficiently rich, even as compared, e.g., to an ER (entity-relationship) model. Even worse, the semantics of the RDF Schema language is not well defined and it is therefore, for example, not always possible to decide whether a given RDF instance corresponds to a given schema. Also certain forms of reasoning, e.g., exploiting subClass or subProperty relationships, are not supported as there exists no formally defined semantics for this type of reasoning.

This led to the development of ontology languages that satisfy the following three requirements:

• They must be highly intuitive for the human user. For example, the object-oriented modeling paradigm is highly successful because of it's intuitive nature, thus an ontology language should achieve the same. This must however not be at the cost of sacrificing the expressive power of the language.

• They must have a well-defined formal semantics with established reasoning properties to ensure completeness, correctness, and efficiency in order to enable the automatic derivation of new statements within a given ontology.

• They must be compatible, both syntactically as well as semantically, with existing Web languages such as XML and RDF to ensure interoperability.



The OWL language was developed to match these criteria. It unifies ideas from different earlier developments and fields: rich modeling primitives as used in frame-based systems (a well established logical framework for knowledge-based systems), formal semantics and efficient reasoning support as developed in the context of description logics (a decidable fragment of first order logic), and the existing Web standards XML and RDF.

• **Description Logics** describes knowledge in terms of concepts and role restrictions that are used to automatically derive classification taxonomies. DL provide theories and systems for expressing structured knowledge, for accessing it and reasoning with it. OWL inherits from DL its formal semantics and the efficient reasoning support.

• **Frame-based systems** provide as central modeling primitive classes (i.e., frames) with attributes. These attributes do not have a global scope but are only applicable to the classes they are defined for. A frame provides a certain context for modeling one aspect of a domain. OWL incorporates the essential modeling primitives of frame-based systems.

• **Web standards: XML and RDF.** Provide Web standards for information representation and exchange. OWL uses these standards as follows: First, OWL has a well-defined syntax in XML based on a document type definition. Second, OWL is an extension of RDF and RDFS.

OWL as RDF Extension

- The language to specify OWL models is given as an RDF Schema
  - like the RDF schema language is expressed within an RDF schema
  - therefore OWL models can be expressed in RDF
  - some of the OWL modeling primitives and RDF modeling primitives overlap and are re-used in OWL

```
<owl:Class rdf:ID="herbivore">
   <rdf:type
        rdf:resource="http://www.ontoknowledge.org/#DefinedClass"/>
   <rdfs:subClassOf rdf:resource="#animal"/>
   <owl:disjointWith rdf:resource="#carnivore"/>
</owl:Class >
```


This is a simple example of OWL, expressed in an extended RDF syntax. In OWL, as in RDFS, classes are defined to categorize resources and classes can be subclasses of other classes. The class "**herbivore**" shows however an example of the richer expressivity of OWL. This class is not only defined as subclass of "**animal**", which is also possible in RDFS, but expresses other set-theoretic properties of the class, such as disjointness with other classes. Such complex class specifications, supporting the basic set-theoretic operators of intersection, union and complement, are one example of how OWL extends the expressivity of RDFS. A big advantage of designing OWL as an extension of RDFS is that RDFS documents are fully compatible with the OWL standard, and thus can be supported by tools for processing OWL.

OWL Semantics

- The semantics of OWL is given in *Description Logics (DL)*
  - Description Logics is a fragment (a sublanguage) of first order predicate logic
  - reasoning (deriving/proving statements) can be done (more) efficiently than in first order predicate logic (FOL)
- Example

OWL expression	**disjointWith** herbivore, carnivore
DL expression	herbivore $\subseteq \neg$ carnivore
FOL expression	$\forall x$(herbivore (x) $\rightarrow \neg$ carnivore(x))

- Using FOL reasoning (and thus DL reasoning) we could obtain, for example

  if herbivore $\subseteq \neg$ carnivore, and lion $\subseteq$ carnivore
  then lion $\subseteq \neg$ herbivore

An essential aspect of OWL is its formal semantics. It is given by the theory of Description Logics (DL). Without discussing description logics in detail, we sketch the principle aspects of DL:

Description Logic is a fragment of first order logic (FOL) that has been developed with two objectives: first, it should enable reasoning about classes of objects and their relationships, and, second, reasoning in description logic should be decidable and efficient, i.e., it should be possible to provide an algorithm, that decides for every statement in DL whether it is true or false.

DL is based the two basic constructs of classes and properties. The syntax of DL is an abbreviated syntax of FOL.

We give first an example of how classes are represented in DL, and provide the corresponding expression also in FOL, in order to better understand the intended meaning. The example represents the OWL expression given in an earlier example, defining the class “carnivore” in DL. Since DL is a fragment of FOL, FL reasoning can also be applied to DL statements. An example of a statement that can be derived is given below (which can be easily explained using set-theoretic reasoning). This type of reasoning allows us to derived new statements from a given set of OWL expressions, which then can be exploited to interpret the semantics of datasets that are annotated using OWL.

Property Semantics in OWL

- Classes in OWL (RDF) are *unary* predicates
- Slot constraints (= properties) in OWL are *binary* predicates

OWL expression	**class** herbivore **subClassOf** animal **restriction onProperty** eats **allValuesFrom** plant
DL expression	herbivore $\subseteq$ animal, $\forall$ eats.plant
FOL expression	$\forall x(\text{herbivore}(x) \leftrightarrow \forall y(\text{eats}(x, y) \rightarrow \text{plant}(y))$ $\forall x(\text{herbivore}(x) \rightarrow \text{animal}(x))$

Here we show of how properties are represented in DL. Whereas classes correspond to unary relations in logics, DL properties, such as "eats" correspond to binary relations, as can be seen from the representation of the statement in FOL. DL expressions (also called DL axioms) can specify additional properties of DL properties, such as the property "restriction onProperty eats allValuesFrom plant". These properties of DL properties correspond to universally quantified statements in FOL.

Question

- rdf:type would be represented in first order logic
  1. As a unary predicate
  2. As a binary predicate
  3. It depends on the properties of the class it is applied to

OWL Class Constructors

Constructor	DL Syntax	Example
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human ⊓ Male
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor ⊔ Lawyer
complementOf	$\neg C$	¬Male
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{john} ⊔ {mary}
allValuesFrom	$\forall P.C$	∀hasChild.Doctor
someValuesFrom	$\exists P.C$	∃hasChild.Lawyer
maxCardinality	$\leqslant nP$	⩽1hasChild
minCardinality	$\geqslant nP$	⩾2hasChild

This list provides an overview of all possible class constructors available in OWL.

OWL Axioms (Properties)

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq \neg$Female
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{President_Bush} $\equiv$ {G_W_Bush}
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{john} $\sqsubseteq \neg${peter}
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
inverseOf	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+ \sqsubseteq$ ancestor
functionalProperty	$\top \sqsubseteq \leqslant 1P$	$\top \sqsubseteq \leqslant$1hasMother
inverseFunctionalProperty	$\top \sqsubseteq \leqslant 1P^-$	$\top \sqsubseteq \leqslant$1hasSSN$^-$

These are examples of predicates (used to state axioms) that can be given in OWL.

OWL sublanguages

- OWL lite
  - Subset of OWL (Full)
  - Easy to implement
- OWL DL
  - Restriction to FOL fragment
  - Well-defined semantics (Description Logic)
  - Mixing of RDFS and OWL restricted
  - Disjointness of classes, properties, individuals and data values
  - Well-defined computational properties
- OWL Full
  - Union of OWL syntax and RDF
  - No restrictions

OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full. OWL Lite is designed for easy implementation and to provide users with a functional subset that will get them started in the use of OWL. OWL DL (where DL stands for "Description Logic") is designed to support those parts of OWL that correspond precisely to Description Logics and to provide a language subset that has desirable computational properties for reasoning systems. The complete OWL language (called OWL Full to distinguish it from the subsets) extends OWL DL so as to make available features, which may be of use to many database and knowledge representation systems, but which go beyond DL.

Question

- Which is true?
  1. For every OWL expression there exists an equivalent first order logic expression
  2. For every first order logic expression there exists an equivalent OWL expression
  3. Neither of the two statements is correct in general

New Technical Problems in Processing Web Data

- Web data management has to deal with challenges different from conventional data management (e.g. in business)
  - Syntactic and semantic heterogeneity
  - Data exchange
  - High degree of autonomy of data sources
- Web data management is based on standards that define data models with properties different from conventional data models (e.g. relational)
  - Schema-less data
  - Optional schema elements
  - Tree- and graph-oriented data models
- Web data management requires new techniques that are able to deal with these new data models and specific processing challenges
  - Storage: storing tree- and graph structured data
  - Indexing: for path-oriented access and both push and pull based systems
  - Schema extraction: for efficient access and understanding of semi-structured data

After this overview on the basic models that are used in Web data management we will in the following investigate some of the resulting data processing challenges and techniques.

Summary

- What is the difference between derived and associative metadata ?
- How is an RDF statement structured ?
- Which are the three views of RDF statements ?
- What is reification ?
- How is an RDF statement represented in XML ?
- What can be specified with RDF schema ?
- Which inheritance mechanisms does RDF schema provide ?
- Which are the reasons that RDF is not considered as being suitable as an ontology language ?
- Which are the requirements an ontology language should satisfy ?
- Which are examples of extensions in expressive power OWL introduces when compared to RDFS ?
- Which approach is used to provide a precise semantics for OWL (and thus implicitly also for RDF) ?

References

- WebSite
  - XML, XPath, Xquery, RDF: http://www.w3.org/
  - OWL: http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1.1
  - WordNet: http://www.cogsci.princeton.edu/~wn/
- Book
  - S. Abiteboul, P. Bunemann, D. Suciu: Data on the Web: From Relations to Semistructured Data and XML, Morgan Kaufman, 2000.
- Articles
  - Grigoris Antoniou, Frank van Harmelen, Semantic Web Primer, MIT Press, 2nd edition, 2004.
  - Jeen Broekstra, Michel C. A. Klein, Stefan Decker, Dieter Fensel, Frank van Harmelen, Ian Horrocks: Enabling knowledge representation on the Web by extending RDF schema. WWW 2001: 467-478
  - Stefan Decker, Sergey Melnik, Frank van Harmelen, Dieter Fensel, Michel C. A. Klein, Jeen Broekstra, Michael Erdmann, Ian Horrocks: The Semantic Web: The Roles of XML and RDF. IEEE Internet Computing 4(5): 63-74 (2000)