The Web has generated a new class of data models, which are generally summarized under the notion semi-structured data models. The reasons for that are manifold: the ubiquity of documents and messages, the extensive use of the hypertext paradigm, the lack of schema information. In this chapter we want to explore their properties and techniques required to deal with them.

In the first part of this chapter we want to introduce the most important semi-structured data model on the Web, XML. Then we will, in the following parts study data management methods for these new data models, and then take a more abstract view by studying graph-oriented semi-structured data models, their close relationships with finite automata and applications of automata theory for indexing and structure extraction. Finally we will investigate their use for semantically annotating and integrating information sources on the Web.
Today’s Questions

1. What is XML?
2. XML Syntax
3. XPath and XQuery
4. What is RDF?
5. RDF Syntax and RDF Schema
Data on the Web: XML

• Limitations of HTML
  - Structure of data expressed as layout
  - Semantics of data hard to analyse and difficult to share
  - No schemas, no constraints

• Thus XML (eXtensible Markup Language) has been developed
  - Markup language to define structured documents
  - Document schemas to fix the structure of documents
  - User-defined markup to express semantics
  - XML architecture for processing and extended functionality

Eventually data too was represented in HTML (tables, lists, matrices). It quickly turned out that this was not a good idea as it became hard to interpret the data correctly, not only for humans but even more for computers. The problem was that this lacked the good old wisdom that you should define a schema according to which your data is structured, before you populate a database. Therefore XML was invented. It transferred the merits of schemas into the document world. Document-oriented people also like to say that they allow « user-defined » markup, as opposed to the « system markup » of HTML.
Example: 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>6 Oak Avenue</street>
    <city>Old Town</city>
    <state>PA</state>
    <zip>95819</zip>
  </billTo>
  <items>
    <item partNum="872-AA">
      <productName>Lawnmower</productName>
      <quantity>1</quantity>
      <price>149.95</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>
```

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.
Example: XML - as data model

2002-13-12

orderDate

purchaseOrder

shipTo

billTo

comment

items

orderDate

name

street

city

state

zip

AliceSmith

Mill Valley

CA

90952

123 Maple Street

billTo

name

street

city

state

zip

Hurry ...

items

partNum

productName

quantity

price

comment

124-AA

1

152.99

confirm order

Lawnmower

©2002, Karl Aberer, EPFL-SSC, Laboratoire de systèmes d'informations répartis
Data and Documents

• Serialization

Document = medium for exchange of information

Information system 1

Communication

Information system 2

©2002, Karl Aberer, EPFL-SSC, Laboratoire de systèmes d'informations répartis
In addition, XML has a kind of schema language, XML Document Type Definitions (DTDs). They specify WHICH element tags may be used and HOW they can be used.
The importance of XML lies not only in its inherent properties as document markup language, but also in the fact that it is used as the basis for a whole architecture covering almost every aspect of information processing imaginable. This architecture is developed under the auspices of the W3C consortium and builds on XML 1.0 which has been approved by the W3C. The architecture comprises the following components:

Programming API’s: DOM and SAX, these are required to build applications that can process XML documents that have been parsed and made available in main memory.

Layout and hyperlinks: standardize the language constructs needed in order to represent the hypertext and layout properties of documents.

Programming APIs, Layout and Hyperlinks are covered in the lecture « Conception of Information Systems »

XML Data Management: XML Schema and Query provide the languages for managing large data-oriented documents and document-collections.

Metadata: as XML data will be exchanged and processed in many different places, it will often be necessary to provide additional data (information) about the data (metadata) in order to enable applications and users to correctly interpret and use the data.

We will learn in this lecture in detail about data and metadata management.
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 Hypermedia documents on 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 (partially structured) data

- XML and Distributed Information Systems
  - XML as data model for managing semi-structured data
  - XML as canonical model to integrate heterogeneous data
  - XML as canonical data format to exchange data among information systems

- Tim Bray:
  "XML will be the ASCII of the Web - basic, essential, unexciting"

The interpretation of what XML is depends also on the viewpoint one takes: in the simplest case it may be considered as a document syntax, but depending on the context other interpretations are possible. It is a language to describe hierarchically structured text documents (this is also where the origin of XML lies, there was an earlier very similar language SGML from which XML was derived, and that was used by the publishing industry). By the W3C XML was chosen to be the foundation of the whole architecture of the Web, which explains also why it is called successor of HTML. For database persons XML is another way of how to structure data, with the additional advantage of getting a textual representation (or a serialization as it is often called) for free. The textual form of a XML document can thus also directly be used to exchange data via messages, which is why the EDI (Electronic data Interchange) community views XML nowadays as their standard message syntax. From the data perspective XML also lead to a more fundamental change in the way data management is perceived, namely the focus on the new question of how to manage, of which we have no or only partial knowledge of how it is structured. We will dedicate part 3 of this chapter mostly to answer this question.
Summary

• What is XML?

• What is the difference between XML and HTML?

• What is the difference between the XML syntax, architecture and applications?

• What is the difference between data and document?

• What are applications of XML?
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 XML supports schemas (by means of DTDs) XML documents are not required to have ones. Rather any document that following some basic syntax concerning the tags are valid XML documents. These are called well-formed XML documents. The constituents of an XML documents are elements (the tags), attributes and textual content. The tags must be nested, such that the elements form a hierarchy. Therefore it is always possible to view an XML document also as a tree, as illustrated by the example. 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 (trees are a kind of data structure!)
Well-formed XML can be specified by means of a formal syntax of which we illustrate only some important production rules. In particular one can see the hierarchical buildup for the production of element. This syntax implies some properties that well-formed XML documents have. However, there exist more requirements on well-formed XML, which are outside the scope of the syntax.
Examples

non-XML:
<shipTo>
 Alice
</shipTo>
<billTo>
 Bob
</billTo>

non-XML:
<1-shipTo>
 Alice
</1-shipTo>

non-XML:
<shipTo>
 Alice
</shipTo>
<billTo>
 Bob
</billTo>

XML:
<shipTo country="US"/>

non-XML:
<shipTo>
 Alice
</shipTo>
<billTo>
<shipTo country="US"/>
 Bob
<shipTo country="CH"/>
XML document type definitions (DTDs) are used to specify
-which elements and attributes are allowed
-And how they can appear in the document in relation to each other
However, in difference to database schemas they are not given as types (a relational schema is for example a (data) type definition) but rather as a special kind of grammar.
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>`

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 are other elements, text, which is represented by special built-in element name #PCDATA (=parseable character data), the empty content or any content, which imposes no further constraints.
From the atomic content model one can construct 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 allow documents 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 heritage from the document processing world, 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.
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 is one case where attributes’ expressive power goes beyond that of elements. It allows 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 allows to factor out repeating parts in the documents and document type definitions.
Usage of DTDs

External DTD Declaration

```xml
<?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

```xml
<!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.
Summary

- Which syntactic and semantic constraints are imposed on a well-formed XML document?

- Which atomic element types and operators can be used to build an element content model?

- Which default values exist for attributes?

- Is a DTD always a separate document?
The problem to address is the following: given an XML document, how can we answer queries as the following: “find the price of all items in the purchaseOrder with partNum “124-AA”. For relational databases this would be a typical query to be expressed in SQL. We introduce now the corresponding counterparts for XML.
Example XPath Query

```
purchaseOrder[orderDate="2002-13-12"]
```

```
/purchaseOrder/items/item[@partNum="124-AA"]/price
```

©2002, Karl Aberer, EPFL-SSC, Laboratoire de systèmes d'informations répartis
Example XPath Query

```
purchaseOrder
    orderDate: 2002-13-12
    items
        item
            partNum: 124-AA
            productName
                quantity
            comment
            price
            item
                partNum
            comment
            price
    billTo
        name: AliceSmith
        street: 123 Maple Street
        city: Mill Valley
        state: CA
        zip: 90952
    shipTo
        name: AliceSmith
        street: 123 Maple Street
        city: Mill Valley
        state: CA
        zip: 90952

context
    /purchaseOrder/items/item[@partnum="124-AA"]/price

filter
    location step
    result

absolute location path

©2002, Karl Aberer
```
The simple navigation pattern in the previous example (downwards to elements with specific names) is generalized in XPath. Each navigation step is characterized by three things:

The direction, i.e. a navigation needs not necessarily move from a parent to a child node, but can follow any relation among elements, in particular by traversal of elements at the same tree level according to the document order (following, preceding) and traversal of multiple nodes (descendants). A complete account of the possible navigation operators is given. The node test checks whether a element node that is encountered in the navigation matches a certain element name. And finally the predicates are the filter expressions which allow to select nodes based on other properties than their name. In particular, filters allow to use other XPath queries to check a property of an element. In that case the predicate is considered as successfully evaluated if a non-empty result set is generated. In order to take account of the different axis operators an extended syntax is used in XPath that specifies for each location step the axis operator. In practice however, the abbreviated syntax that we have already seen is more common.
Example

2002-13-12

orderDate

purchaseOrder

billTo

shipTo

comment

items

orderDate

Example

2002-13-12

orderDate

purchaseOrder

billTo

shipTo

comment

items

item

item/price

./item

item/..

*/

item/0*

item[1]

item[1,4]

item/price

item/productName="Lawnmower"

item[@partNum=124-AA]

/purchaseOrder/item

/purchaseOrder/*/item

/purchaseOrder/billTo/items/item

/purchaseOrder/items/item

/purchaseOrder/billTo and shipTo

Hurry ...

124-AA

partNum

productName

quantity

price

comment

Lawnmower

1

152.99

confirm order
By now it should have become clear that 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 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. We introduce now the most important additional concepts of XQuery as they were specified in June 2001. The goal is not to obtain a thorough knowledge and capability to use Xquery, but to understand the substantial additional concepts to Xpath and the relationships to SQL. We must be aware that this standard is not yet finalized and we may expect certain changes (probably not major ones).

With a knowledge of Xpath and SQL the following presentation of Xquery should be straightforward to follow.
Element Constructor and FLWR Expressions

- `<result>`
  
  ```
  LET $a := \text{avg(document("bib.xml")//book/price)}$
  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 exhibits a whole wealth of new concepts of 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 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 new a WHERE clause is available to express conditions. This allows in particular to express joins when multiple variables are bound in the FOR clause. The third observation is that a RETURN clause allows 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.
The semantics of Xquery expressions is defined similarly to SQL (which is in short: build the Cartesian product of the relations in the FROM clause, evaluate the predicates in the WHERE clause and then project on the attributes in the SELECT clause). Also for FLWR expression first generate all tuples from the Cartesian product space of all sets to which variables are bound. An important difference that the order among document elements needs to be preserved, therefore also the order in which the variables appear in the FOR clause has an impact on the order the result tuples will be sorted. The WHERE clause is evaluated as for SQL and for each remaining tuple an XML document fragment is generated by replacing the variables by the tuple values. There is also a XML query algebra under development which is intended to provide a precise semantics to Xquery. As all these are ongoing developments we will not further delve into the technical details, as the final outcome of these developments is not yet determined.
<book year="1989">
<title>TCP/IP Illustrated</title>
<author>Clifford B. Rushforth</author>
<publisher>Addison-Wesley</publisher>
<price>$19.95</price>
</book>

<book year="1992">
<title>Advanced Programming in the Unix environment</title>
<author>Clifford B. Rushforth</author>
<publisher>Addison-Wesley</publisher>
<price>$49.95</price>
</book>

<book year="2000">
<title>Data on the Web</title>
<author>Abiteboul</author>
<publisher>Morgan Kaufmann Publishers</publisher>
<price>$39.95</price>
</book>

<book year="1999">
<title>The Economics of Technology and Content for Digital TV</title>
<editor>Gerhard</editor>
<publisher>Kluwer Academic Publishers</publisher>
<price>$129.95</price>
</book>

©2002, Karl Aberer, EPFL-SSC, Laboratoire de systèmes d'informations répartis
Q1

List books published by Addison-Wesley after 1991, including their year and title.

Solution in XQuery:

```xml
<xquery>
  <bib>
    <book year="1994">
      <title>TCP/IP Illustrated</title>
    </book>
    <book year="1992">
      <title>Advanced Programming in the Unix environment</title>
    </book>
  </bib>
</xquery>
```

Expected Result:

```xml
<xquery>
  <bib>
    <book year="1994">
      <title>TCP/IP Illustrated</title>
    </book>
    <book year="1992">
      <title>Advanced Programming in the Unix environment</title>
    </book>
  </bib>
</xquery>
```
Q2

Create a flat list of all the title-author pairs, with each pair enclosed in a "result" element.

Solution in XQuery:

```xquery
<x-query>
  for $t in document("http://www.bn.com"/bib/book,
    {$t in $t/title,
      {$a in $a/author
        TITLES
        <result>
          {$t}
          {$a}
        </result>
      }
    }
  )
</x-query>
```

Expected Result:

```xml
<result>
  <title>TCF/IF Illustrated</title>
  <author>McGraw/Hill</author>
</result>
```

©2002, Karl Aberer, EPFL-SSC, Laboratoire de systèmes d'informations répartis
Q3

Solution in XQuery:

```xml
<xquery>
  <result>
    <title>TCP/IP Illustrated</title>
    <author>Stevens</author>
  </result>
  <result>
    <title>Advanced Programming in the Real Environment</title>
    <author>Stevens</author>
  </result>
</xquery>
```

Expected Result:

```xml
<xquery>
  <result>
    <title>Advanced Programming in the Real Environment</title>
    <author>Stevens</author>
  </result>
  <result>
    <title>TCP/IP Illustrated</title>
    <author>Stevens</author>
  </result>
</xquery>
```

how is this query different from Q2?
Q5

For each book found at both bn.com and amazon.com, list the title of the book and its price from each source.

Solution in XQuery:

```xquery
<books-with-prices>
  for $b in document("www.bn.com/bib.xml")//book,
  $a in document("www.amazon.com/prices.xml")//entry
  where $b/title = $a/title
  return <book-with-prices>
    <title>{$b/title}</title>
    <bm-price>{$b/price/text()}</bm-price>
    <am-price>{$a/price/text()}</am-price>
  </book-with-prices>
</books-with-prices>
```

**Expected Result:**

```xml
<books-with-prices>
  <book-with-prices>
    <title>Java 2D in Action</title>
    <bm-price>65.95</bm-price>
    <am-price>65.99</am-price>
  </book-with-prices>
  <book-with-prices>
    <title>Advanced Programming in the Unix Environment</title>
    <bm-price>65.95</bm-price>
    <am-price>65.99</am-price>
  </book-with-prices>
</books-with-prices>
```

©2002, Karl Aberer, EPFL-SEE, Laboratoire de systèmes d'informations répartis
Functions

- NAMESPACE
  xsd=http://www.w3.org/2001/03/XMLSchema-datatypes

  FUNCTION depth(ELEMENT $e) RETURNS xsd:integer
  {
    -- An empty element has depth 1
    -- Otherwise, add 1 to max depth of children
    IF empty($e/*) THEN 1
    ELSE max(depth($e/*)) + 1 }

  depth(document("partlist.xml"))

  • Find the maximum depth of the document named "partlist.xml"

Xquery goes much further in terms of expressiveness than SQL by allowing the definition of arbitrary user-defined functions. This is a feature that can be found in SQL99 (as well), the object-relational query language standard building on SQL92. We also see in this example, that Xquery uses type specifications that are provided in the XML Schema standard, which too is currently under development.
Existential and Universal Quantifiers

- FOR $b$ IN //book
  WHERE
    SOME $p$ IN $b$/para SATISFIES contains($p$, "sailing")
    AND contains($p$, "windsurfing")
  RETURN $b/title

  • Find titles of books in which both sailing and windsurfing are mentioned in the same paragraph.

- FOR $b$ IN //book
  WHERE
    EVERY $p$ IN $b$/para SATISFIES contains($p$, "sailing")
  RETURN $b/title

  • Find titles of books in which sailing is mentioned in every paragraph.

As in SQL, Xquery also supports the concepts of universal and existential quantification of variables ranging over set expressions.
Q4

For each author in the bibliography, list the author's name and the titles of all books by that author, grouped inside a "result" element.

Solution in XQuery:

```
<result>
  for $a in distinct-values(document("http://www.ibs.com")//author)
  return
    { $a | 
        where some $ta in $b/author satisfies deep-equal($ta,$a)
        return $b/title
    }
</result>
```

XPath expression returns exactly all different authors

In the above query, `deep-equal()` tests to see if the two nodes have the same structure and values.

Expected Result:

```
<results>
  <result>
    <author>
      <last>Stevens</last>
      <first>Mike</first>
    </author>
    <title>CFI: Illustrated</title>
  </result>
</results>
```

©2002, Karl Aberer, EPFL-SSC, Laboratoire de systèmes d'informations répartis
Summary

• What is the relationship between XPath and XQuery?
• Which result types are possible in XPath?
• What is an axis in XPath?
• What is the difference between a/*/b and a//b?
• How can a join operation be specified using XQuery?
• What are differences and commonalities among XQuery and SQL92?
The notion of metadata is very simple to understand:
- Metadata is data about data
- Metadata is required to process data correctly
Examples:
- We need a relational database schema (metadata) in order to pose correct queries, check correctness of queries etc.
- We need metadata on documents to properly interpret its contents, e.g. which language the document is written in, in order to process the textual content, the version number in order to find the most recent document, etc.

We can distinguish two fundamentally different kinds of metadata. Derived metadata is an abstraction of the data itself and can (in principle) be mechanically checked against the data. The typical examples for these are database schemas and constraints. We will later see of how such metadata an be derived from XML-like data (we will call it semistructured data) in case it has not been provided a priori. Associative metadata is data that is provided by some outside agency. ‘Outside’ can mean it is either a human user or another computer system that we do not consider as part of the existing context (e.g. a tool to analyze textual content). In the following we will first look at how associative metadata can be represented in the framework of XML. The current proposal for doing this is RDF, a mechanism to associate metadata with XML documents.
Requirements on Associative Metadata

- Metadata represents knowledge
- Knowledge on the Web is distributed
  - link knowledge on the Web
- Many different user communities
  - Extensibility and simplicity
    --> Resource Description Framework (RDF)
- Knowledge on the Web is biased
  - there is no universal truth
  - it must be possible to reason about statements

For the Web there exists these main requirements on metadata:

1. Knowledge is distributed and must therefore be made explicit and available to others in order to allow to connect knowledge and derive from it far reaching conclusions than is possible to get from local knowledge only.

2. In order to be able to exchange knowledge in the open Web environment it must be very simple to represent it, and it must be easy to extend the mechanisms that are used to represent knowledge.

These are the two major concerns RDF addresses. Once knowledge is made available it must also be able to reason on it. This is today a still far-reaching goal which is however a topic of intense research. In the following we concentrate on the first two questions, how a simple and extensible framework to represent knowledge on the Web looks like. RDF is at the current time the prevailing mechanism in discussion that is considered for knowledge/metadata representation, but time has to show how well and successfully it will be adopted in practice. Nevertheless for the purpose of being able to follow the ongoing developments it is necessary to understand what is technically feasible today.
The RDF is a standard supported by the W3C (http://www.w3.org/RDF/) to represent associative metadata. It is a fairly simple, graph-oriented data model to annotate any kind of XML document.

RDF currently plays a major role in the discussions on the Semantic Web: this is Tim Berner Lee’s new vision (and currently supported by many funding agencies) that to make real use of the information that is available on the Web, it must become semantically enriched, in short, more (associative) metadata is required in order to provide sufficient context for meaningful processing of the data. (Machine-understandable Semantics)

A key idea of XML was the separation of presentation from structure. With RDF a next step is taken by separating semantics from structure. This would allow to use common semantic descriptions for different structural representations.

Typical applications include PICS (annotating documents with information on the suitability of the content for certain groups, e.g. like the movie rating system), Dublin Core (annotating documents with basic bibliographic information. Anecdote: this is an effort from the Digital Library community and was initiated in a city called Dublin, which is not located in Ireland but in US), and site and topic maps that provide subject taxonomies (concept hierarchies) for documents. /.

RDF Resource Description Framework - Applications

- Data model for representing associative metadata
- Semantic Web
- Some applications
  - PICS: Platform for Internet Content Selection (Rating, Censoring)
  - Dublin Core: Author, Subject, Date, ... of Internet Resources (DLs)
  - Site Maps: Subject Taxonomies
  - Topic Navigation Maps: Multidimensional Taxonomies
RDF Resource Description Framework

• RDF (Instances)
  - Statements about resources (addressable by an URI) and literals (strings)
  - Statements are of the form: subject predicate object
  - Like simple natural language sentences

• RDF-Schema
  - Data model to specify schemas for RDF instances
  - Which predicates are applicable for which objects with which subjects
  - "grammar" and "vocabulary"
RDF Data Model

- **Resources**
  - A resource is a thing you talk about (can reference)
  - Resources have URI's
  - RDF definitions are themselves resources (see requirements)

- **Properties**
  - define relationships to other resources or atomic values

- **Statements**
  - "Resource has Property with Value"
  - Values can be resources or atomic XML data

---

/. RDF consists of two parts: a language for the metadata instances (RDF), which allows to connect simple « sentences » with document parts that are addressed by Universal Resource Identifiers (URI – we will call these document parts in the following resources), of which URL’s are the most important example. And a language for specifying schemas for RDF Instances, that defines the possible vocabularies and the grammar of how these sentences may be formed. So the situation is very similar as with well-formed XML (instances) and XML-DTD (schemas).

It is important to keep in mind that RDF is currently only specified in a « semi-formal » manner, that means the XML Syntax is well specified, however the underlying mathematical abstraction, based on graphs, is only informally described. Also very few tools (e.g. for querying RDF data) exist at the current time and it is in fact not yet very clear which kind of methods will be required to process RDF data (developments are on providing mechanisms for logical reasoning, parsing, storing and querying RDF data).

The RDF model is in some sense similar to the ER model. It knows Entities, which are called resources, and relationships, which are called properties. The main difference is that RDF requires that relationships are principally directed, and carry the semantics that 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).
We can view RDF instances in three different ways: we can look at them as sentences, where the subject and the object can be either URI’s or Strings, we can view them as directed graphs, where the subject and object are considered as nodes and the predicate is considered as link, or we can view them as XML documents where the RDF metadata is represented as an XML document. Depending on the purpose (simply understanding the meaning, processing and algorithms for the annotation structure, or representation for exchange and storage) one of the forms is chosen.
Simple values for the objects of an statement are just the simplest case. In general if we have a complex object about which we want to make a statement we create a new resource that becomes the object. We give here only the representation in the graph model. We will discuss the XML serialization after we have introduced the various constructs of RDF.
Sometimes we want to make annotations not to a resource part but to whole set of resource. RDF provides a mechanism to do that also. It introduces the concept of containers, which are sets of resources. Three types of containers are distinguished: bags which represent unordered multisets (= sets with multiple occurrences of the same resources), sequences which represent ordered sets of resources and alternatives which represent a single resource that is to be chosen out of a given set.

When annotating containers there is subtlety that needs to be taken into account: two different meanings could be associated with such an annotation: either the annotation could refer to the container itself (e.g. in the example Urs has collected this set of slides but not authored them, thus he is only the author of the collection), or the annotation could refer to each single member of the collection (e.g. Urs would be the author of all slides).

In the graph representation of RDF care has to be taken to reflect correctly the order of the elements in a container, therefore a number index is used for labelling the links to the members.
Next we look at a point which is a bit intricate. It is sometimes not only necessary to annotate collections of data, but to annotate annotations as well. After all metadata itself is also data, and we can as well have metadata on metadata and so forth. In order to be able to do that, we have to turn an RDF statement into a resource that can be annotated.

We look at this problem first in the graph representation. In order to be annotatable, something must be a node. Therefore we have first to create a node for the statement to be created. Of course, we need to know what this node represents, at least which statement it represents. Therefore we have links to the subject and object of the statement that we want to annotate. In addition we identify the predicate by introducing an extra node (there could be different links between www.doc.ch and Urs Giger). As in the case of collections we have also to identify the type of the new node that we have introduced, i.e. say that it is a statement. Having done all this, we can now annotate the statement.
RDF Syntax

- Many syntactic varieties possible
- Basic form

```
<rdf:RDF>
  <rdf:Description about="Person://1234/1">
    <s:Creator>Urs Giger</s:Creator>
    <s:createdWith rdf:resource="http://www.w3c.org/amaya"/>
  </rdf:Description>
</rdf:RDF>
```

Up to now we have considered RDF statements in their graph representation (this is similar to interpreting an XML document as a tree). As of course the goal is to represent metadata in an exchangeable document format, and consequently in XML Syntax, an appropriate encoding of RDF graphs in XML has to be established. The basic pattern to do this is as follows: the subject is referenced in an 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 elements, e.g. `s:Creator`. The content of this element in turn is the object of the statement. If the object is not a literal, one can alternatively represent the object as an attribute of the predicate element.
Having introduced RDF we are in a similar situation as after introducing well-formed XML. What kind of properties are used to annotate resources is not determined. For that purpose a schema is required. In the context of knowledge/metadata management a « schema » is called ontology. An ontology is a globally agreed vocabulary (possibly together with knowledge on rules/constraints) for a specific application domain. In order to specify schemas for RDF (resp. ontologies) RDF schemas are introduced.
RDF Schema Base Constructs

- rdfs:Resource
  - Everything that can be described

- rdfs:Class
  - Categories for objects and subjects
  - Used to describe which properties a resource belonging to a class has

- rdfs:Property
  - Everything that can connect resources (predicates)

- rdf:type
  - Assigns an RDF instance to its class
  - Default class is rdfs:Resource
  - The RDF instance must have the properties that are declared for the class

RDF Schema provides the constructs that are necessary to describe types of RDF instances. To that extent all concepts that occur in RDF are made available in the form of XML elements names. The tag rdf:type we have already used, is in order to describe the type of a reification statement.
RDF Schema: Declaring Properties

- `rdfs:domain`
  - Constrains the classes of which the instances may have a property

- `rdfs:range`
  - Constrains the classes of which the instances may be the value of a property

- Properties are not declared within class definitions (as in OO programming) but are independent objects which refer to the classes the belong to (as domain or range)

RDF schemas bear a lot of similarity with object-oriented schemas (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.
This example shows a simple RDF schema. A class is declared which serves as domain for a property. In this example we reuse a class definition for Integer that has been defined elsewhere.
RDF Schema Inheritance

- **rdfs:subClassOf**
  - A subClassOf B: Every instance of A is also instance of B
  - transitive, not reflexive (no cycles!), anti-symmetric
  - 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: Irma has Property Father with value Urs, and Father subProperty AParent implies Irma has Property AParent with value Urs

Similar to object-oriented modelling, RDF too provides inheritance mechanisms in order to support the reuse of specifications. RDF Schema provides both for type inheritance and value inheritance. With type inheritance the subtype (in RDF schema called a subclass) has all the properties that its supertype possesses. This means there exists a propagation of properties at the type level. With value inheritance a « subProperty » within an instance inherits the values of the « superProperty ». This means with value inheritance there exists a propagation of properties and the instance/value level.
Finally, even RDF schema is described within an RDF schema. In such a situation we speak of a Meta-Schema. We see in this Meta-Schema that all subclass relationships end up in rdfs:Resource (i.e. rdfs:Resource is the root of the subclass hierarchy). This is consistent with the RDF semantics that everything is a resource. We see also that the most general category (type) is rdfs:Class
RDF Summary

- RDF is a graph data model with nodes (subjects, objects) and directed edges (predicates)
- RDF Schema allows to define ontologies for specific application areas
  - Names of resources and properties
  - Applicability of properties
  - Class inheritance (of properties)
  - Property inheritance (of property values)
Recapitulation

- 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?
## Comparison XML vs. RDF

<table>
<thead>
<tr>
<th>Aspect</th>
<th>XML</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Document/Instance</strong></td>
<td>Nested String, Sequence</td>
<td>Graph, Sequence</td>
</tr>
<tr>
<td><strong>Main Application</strong></td>
<td>Syntax</td>
<td>&quot;Semantics&quot; Metainformation</td>
</tr>
<tr>
<td></td>
<td>Document content</td>
<td></td>
</tr>
<tr>
<td><strong>Schema</strong></td>
<td>Complex Datatypes (Alternative, Sequence, Repetition) over Strings</td>
<td>(Complex) Objecttypes Resource = Object Property = Relationship</td>
</tr>
<tr>
<td><strong>Inheritance</strong></td>
<td>Not existent</td>
<td>SubClass SubProperty</td>
</tr>
<tr>
<td><strong>Main application of schemas</strong></td>
<td>Document Type Conformance</td>
<td>Vocabulary Specification</td>
</tr>
</tbody>
</table>
References

• WebSite
  - XML, XPath, XQuery, RDF: http://www.w3.org/
  - XQuery Engine: http://demo.openlinksw.com:8391/xquery/demo.vsp

• Book