Semantic Web

3. XML

PD Dr. Matthias Thimm

thimm@uni-koblenz.de

Institute for Web Science and Technologies (WeST)
University of Koblenz-Landau

WeST
People and Knowledge Networks

Overview

- Ontologies and description logics are formal tools for knowledge representation and reasoning
- In order make use of the tools on the Web we need ways to represent knowledge on the Web
- XML is the most popular data format for representing (semi-)structured information (such as “knowledge”)
- We will introduce the syntax of XML, XML Schema, and some applications.

Outline

Outline

XML - Overview 1/3

- XML: eXtensible Markup Language
- Derived from structured text (XHTML $\in$ XML $\subseteq$ SGML)
- Web-Standard (W3C) for exchanging data:
  - XML describes inputs and outputs of many applications (in most cases called: services)
  - Industry created and supported XML standards for applications, communication protocols, service descriptions, etc. (e. g. `www.oasis-open.org` or `www.xml.org`)

XML - Overview 2/3

- *Complementary* to HTML
  - HTML is only one of the applications for XML
  - HTML describes presentation layer
  - XML describes the structure of content/data
- *Extensible*, unlike HTML
  - Users can add new tags, and separately specify how the tag should be handled for display
- *Data modeling*: XML is a data model for semi-structured data

XML - Overview 3/3

XML innovations

- Specify new tags
- Create nested tag structures - hierarchical approach
- Enable to exchange (annotated) data, not only documents
- Tags create content independent of visualization (vs. HTML)

Tags make data (relatively) self-documenting:

```
<bank>
  <account>
    <account_number>A-101</account_number>
    <branch_name>Downtown</branch_name>
    <balance>500</balance>
  </account>
  <depositor>
    <account_number>A-101</account_number>
    <customer_name> Johnson </customer_name>
  </depositor>
</bank>
```

Why do we need XML? 1/3

- Data interchange is critical in today's networked world
- Examples:
  - Banking: funds transfer
  - Order processing (especially inter-company orders)
  - Scientific data
    - Chemistry: ChemML, ...
    - Genetics: BSML (Bio-Sequence Markup Language), ...
  - ...

Why do we need XML? 2/3

- Paper flow of information between organizations is being replaced by electronic flow of information
- Each application area has its own set of standards for representing information
- XML has become the basis for all new generation data interchange formats
- Earlier generation formats were based on plain text with line headers indicating the meaning of fields
  - Similar in concept to email headers
  - Does not allow for nested structures, no standard "type" language
  - Tied too closely to low level document structure (lines, spaces, etc.)

Why do we need XML? 3/3

- Each XML based standard defines what are valid elements, using
  - XML type specification languages to specify the syntax
    - DTD (Document Type Definition)
    - XML Schema
  - Plus textual descriptions of the semantics
- XML allows new tags to be defined as required
  - However, this may be constrained by DTDs
- A wide variety of tools is available for parsing, browsing and querying XML documents/data

Outline

Core idea: Nesting of elements 1/2

- Nesting of data
  - Useful in data transfer
  - Create hierarchical data structures
  - Represent subelements of a larger entity
- For example, elements representing Title and Professor are nested within a Lecture element

```
<Event id="o1">
  <Lecture LNr="5001">
    <Title>Introduction to CS</Title>
    <Prof>
      <pnr>2137</pnr>
      <name>Kant</name>
      <loc>C4</loc>...
    </Prof>
  </Lecture>
  ...
</Event>
```

Core idea: Nesting of elements 2/2

- Nesting is not supported, or discouraged, in relational databases
  - With multiple orders, customer name and address are stored redundantly
  - Normalization replaces nested structures in each order by foreign key into table storing customer name and address information
- Nesting is supported in object-relational databases
- But nesting is appropriate when transferring data
  - External application does not have direct access to data referenced by a foreign key

XML and HTML

- HTML: fixed Tags und Semantics (presentation layer)
- XML: variable Tag Set specific for given application or standard (meta-grammar)
- XML $\subseteq$ SGML (Standard Generalized Markup Language)

HTML:

```
<h1>Event</h1>
  <p>
    <i>Intro CS</i>
    Kant
    <br>Tuesday 16:00
  </p>
  ...
```

XML:

```
<Event id="o1">
  <Lecture LNr="5001">
    <Title>Intro CS</Title>
    <Prof>
      <pnr>2137</pnr>
      <name>Kant</name>
      <loc>C4</loc>...
    </Prof>
  </Lecture>
  ...
</Event>
```

XML syntax 1/3

- XML element
  - Object is defined by a pair of corresponding tags, like <Prof> (opening tag) and </Prof> (closing tag)
  - Content of the element: text and other elements (subelements) included between tags
  - Elements can be nested (no depth restrictions)
  - Empty elements: <Year></Year> can be shortened: <Year/>

Start tag	→	<Prof>
Sub element	→	<pnr>2137</pnr>
Sub element	→	<name>Kant</name>
Sub element	→	<loc>C4</loc>
Text	→	Building location can change
End tag	→	</Prof>

XML syntax 2/3

- Elements must be properly nested
  - Proper nesting:
    `<account>... <balance>... </balance></account>`
  - Improper nesting:
    `<account>... <balance>... </account></balance>`
  - Every start tag must have a matching end tag on the same level (same parent element).
- Improper nesting in HTML may not be harmful:

  ```
  In <i>HTML<b>improper</i>nesting</b>may work
  ```

  may still produce

  In *HTML* **improper nesting** may work

XML syntax 3/3

- XML attribute:
  - Name-value pair inside starting tag of element
  - Tied to a specific XML element
  - Alternative notation to nested tags
  - Element can have multiple attributes, but each occurs only once

```
<Prof loc="C4">
  <pnr>2137</pnr>
  <name>Kant</name>
</Prof>
```

Attributes or sub elements?

- Document view
  - subelement contents are part of document contents
  - attributes are part of markup
- Data representation view
  - . . . unclear and confusing . . .
  - Same information can be represented in different ways, e. g.

```
<prof name="Kant">...</prof>
```

    versus

```
<prof><name>Kant</name>...</prof>
```

- Use subelements for content (objects, . . .) and attributes as identifiers of elements

Namespaces

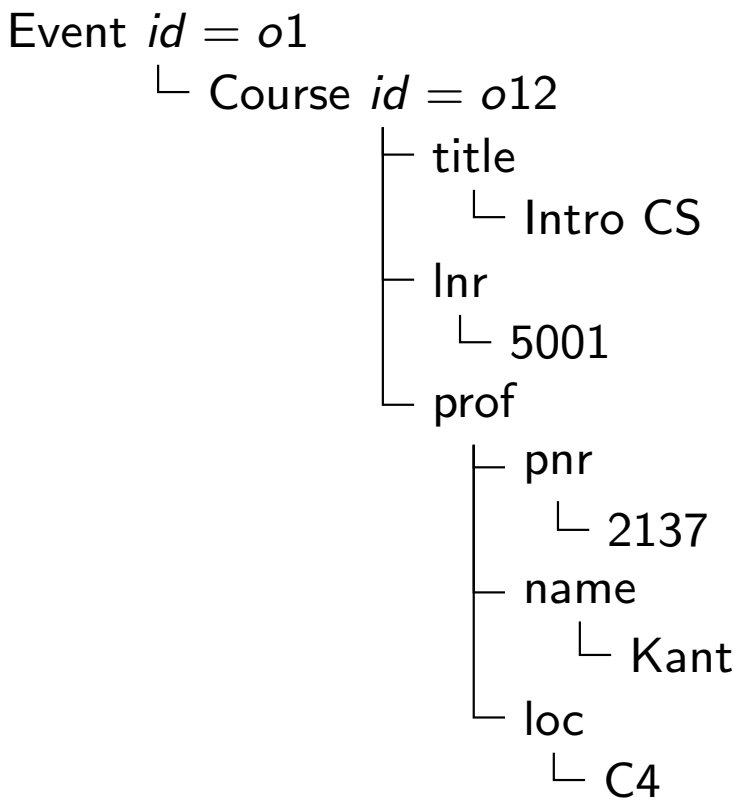
- XML can be exchanged between organizations
- Problem
  - Same tag + Different organizations $\rightarrow$ Different meaning
- Specifying a unique string as an element name avoids confusion
- Better solution: use unique-name:element-name
- Avoid long unique names by using XML Namespaces

```
<Courses xmlns:uni="http://www.university.edu">
  ...
  <uni:lecture>
    <uni:title>Introduction to CS</uni:title>
    <uni:location>F112</uni:location>
  </uni:lecture>
  ...
</Courses>
```

XML model

XML can be represented as a graph (more specifically: as a tree)

```
<Event id="o1">
  <Course id="o12">
    <title>Intro CS</title>
    <lnr>5001</lnr>
    <prof>
      <pnr>2137</pnr>
      <name>Kant</name>
      <loc>C4</loc>
    </prof>
  </Course>
</Event>
```



XML Declaration and DocType

In order to use XML in applications it is necessary that the application knows that it is now reading an *XML document* and which *vocabulary* it uses.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Event SYSTEM "event.dtd">

<Event id="o1">
  <Course id="o12">
    <title>Intro CS</title>
    <lnr>5001</lnr>
    <prof>
      <pnr>2137</pnr>
      <name>Kant</name>
      <loc>C4</loc>
    </prof>
  </Course>
</Event>
```

XML Declaration and DocType

In order to use XML in applications it is necessary that the application knows that it is now reading an *XML document* and which *vocabulary* it uses.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Event SYSTEM "event.dtd">

<Event id="o1">
  <Course id="o12">
    <title>Intro CS</title>
    <lnr>5001</lnr>
    <prof>
      <pnr>2137</pnr>
      <name>Kant</name>
      <loc>C4</loc>
    </prof>
  </Course>
</Event>
```

XML Declaration and DocType

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

- Optional declaration
- Determines XML version to be read and encoding to be used

```
<!DOCTYPE Event SYSTEM "event.dtd">
```

- Defines the structure of the XML (what tags are allowed?)
- References the Document Type Definition (DTD)
- more on that in a minute

Comparison: Relational vs. semi-structured data model

Relational and object model

Pros:

- Clear consistency properties
- Partially: simple and clean formal model

Cons:

- Only pre-defined data structures
- Designed for fully-defined data
- Not interchangeable
- Not easy to read

XML

Pros:

- Easy to read (relatively)
- Incomplete or not fully defined data is not a problem
- Serializable
- Extensible
- Easily interchangeable

Cons:

- No simple and nice model
- Document-centered: not data or object-centric model
- Document can be serialized in different ways

Outline

XML Schemas 1/2

- XML Document:
  - Text Document with XML descriptions
  - Database perspective: XML document is a semi-structured database (includes specific schema)
- Well-formed XML document
  - All Elements are correctly nested with matching start and end Tags
  - Document has one root element
  - It still can contain unstructured text
  - Specific characters in XML have to be represented in special way
- Valid XML Document:
  - Well-formed XML Document, that corresponds to a specific defined XML Schema
  - XML Schema is used to validate document
  - Appropriate for data used in Web Portal

XML Schemas 2/2

- Schemas are very important for XML data exchange
  - Otherwise, a site cannot automatically interpret data received from another site
- Two mechanisms for specifying XML schema
  - Document Type Definition (DTD)
    - Widely used
  - XML Schema
    - Newer, more powerful but more complicated

Document Type Definition (DTD)

- DTD specifies type and structure of XML document
- DTD constraints structure of XML data
  - What elements can occur
  - What attributes can/must an element have
  - What subelements can/must occur inside each element, and how many times.
- DTD does not constrain data types
  - All values represented as strings in XML
- DTD syntax
  - `<!ELEMENT element (subelements-specification) >`
  - `<!ATTLIST element (attributes) >`

Elements in DTD

- Sub elements are specified as
  - names of elements, or
  - #PCDATA (parsed character data), i. e., character strings, or
  - EMPTY (no sub elements) or ANY (anything can be a sub element)
- Sub element specification may have regular expressions
  - Notation
    - "|" : alternatives
    - "+": 1 or more occurrences
    - "∗": 0 or more occurrences
- Example

```
<!DOCTYPE bank [
  <!ELEMENT bank ( ( account | customer | depositor)+)>
  <!ELEMENT account (account_number branch_name balance)>
  <!ELEMENT balance(#PCDATA)>
  <!ELEMENT customer_name(#PCDATA)>
  ...
]>
```

XML Schema

- XML Schema
  - much more expressive than DTD
  - significantly more complicated than DTD
- XML Schema supports
  - Typing of values
    - Integer, string, etc.
    - Constraints on min/max values
  - Complex types (user-defined)
  - Many more features, including
    - Uniqueness and foreign key constraints, inheritence
- XML Schema is
  - Specified in XML syntax
  - More standard representation (but verbose)
  - Already integrated with namespaces

Example of XML Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="bank" type="BankType"/>
<xs:element name="account">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="account_number" type="xs:string"/>
      <xs:element name="branch_name" type="xs:string"/>
      <xs:element name="balance" type="xs:decimal"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
...
<xs:complexType name="BankType">
  <xs:sequence>
    <xs:element ref="account" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="customer" minOccurs="0" maxOccurs="unbounded"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

Usage of DTDs and XML Schema

If `types.dtd` contains a document type definition that it can be used in XML files via

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE bank SYSTEM "types.dtd">

<bank>
...
```

If `schema.xsd` contains XML Schema then

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

<bank xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="schema.xsd">
...
```

Outline

Summary

- XML as a modeling language
  - provides an easy (and standardized) means to represent (semi-)structured information
- XML syntax:
  - Elements
  - Attributes
  - Namespaces
- XML Schema and DTD

Pointers to further reading

- Extensible Markup Language (XML) 1.1 (Second Edition). `http://www.w3.org/TR/xml11/`
- XML Tutorial: `http://www.w3schools.com/xml/`
- XML Validator: `http://validator.w3.org`