NPFL092 Technology for Natural Language Processing

Introduction to XML

Zdeněk Žabokrtský, Rudolf Rosa

November 24, 2020

EUROPEAN UNION
European Structural and Investment Fund
Operational Programme Research,
Development and Education

Charles University
Faculty of Mathematics and Physics
Institute of Formal and Applied Linguistics

eXtensible Markup Language

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

<my_courses>

    <course id="NPFL092">
      <name>NLP Technology</name>
      <semester>winter</semester><hours_per_week>1/2</hours_per_week>
      <department>Institute of Formal and Applied Linguistics</department>
      <teachers>
        <teacher>Rudolf Rosa</teacher>
        <teacher>Zdeněk Žabokrtský</teacher>
      </teachers>
    </course>

</my_courses>
```

Outline

- basic properties of XML
- syntactic requirements
- well-formedness vs. validity
- pros and cons

Markup languages

- a markup language - a set of rules for annotating a text (=adding information into it)
- marks must be syntactically distinguishable from the text (hence, some kind of escaping is always needed)
- markup can specify a formatting of text segments, or their meaning (semantics), or both
- a markup language can be line oriented or not
- typically at least partially "recursive" (a CFG is needed for parsing it)

History

- markup used since 1960s
  - markup = inserted marks into a plain-text document
  - e.g. for formatting purposes (e.g. TEXin (1977
- 1969 – GML – Generalized Markup Language
  - Goldfarb, Mosher and Lorie, legal texts for IBM
- 1986 – SGML – Standard Generalized Markup Language, ISO 8879
  - too complicated!
- 1992 – HTML (Hypertext Markup Language)
  - only basics from SGML, very simple
- 1996 – W3C new directions for a new markup language specified, major design decisions
- 1998 – XML 1.0
- 2004 – XML 1.1, only tiny changes, XML 2.0 not under serious consideration now

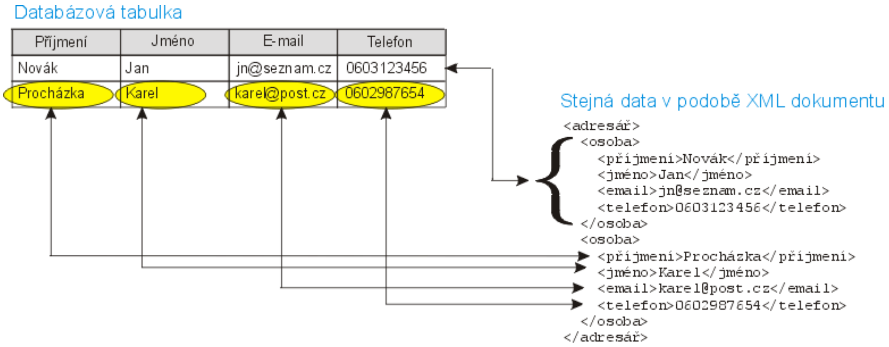
eXtensible Markup Language

- Language – a convention capturing a certain subset of Σ^*; it can be decided whether a string does or doesn't belong to the language,
- Markup – additional information inserted into the text in a form of textual marks, which are, however, distinguishable from the text itself.
- eXtensible – complexity can be scaled up according to your needs (as opposed to, e.g., HTML or markdown, whose mark inventories cannot be changed by users)

Advantages of XML

- open file format, specification for free from W3C (as opposed to some proprietary file formats of database engines or text editors)
- easily understandable, self-documented files
- text-oriented – no specialized tools required, abundance of text editors
- possibly more semantic information content (compared e.g. to formatting markups - e.g. "use a 14pt font for this" vs "this is a subsection heading")
- easily convertible to other formats
- easy and efficient parsing / structure checking
- support for referencing

Relational Databases vs. XML



Databázová tabulka

Příjmení	Jméno	E-mail	Telefon
Novák	Jan	jn@seznam.cz	0603123456
Procházka	Karel	karel@post.cz	0602987654

Stejná data v podobě XML dokumentu

```
<adresář>
  <osoba>
    <příjmení>Novák</příjmení>
    <jméno>Jan</jméno>
    <email>jn@seznam.cz</email>
    <telefon>0603123456</telefon>
  </osoba>
  <osoba>
    <příjmení>Procházka</příjmení>
    <jméno>Karel</jméno>
    <email>karel@post.cz</email>
    <telefon>0602987654</telefon>
  </osoba>
</adresář>
```

Credit: kosek.cz

Relational Databases vs. XML

Relational databases

- basic data unit – a table consisting of tuples of values for pre-defined “fields”
- tables could be interlinked
- binary file format highly dependent on particular software
- emphasis on computational efficiency (indexing)

XML

- hierarchical (tree-shaped) data structure
- inherent linear ordering
- self-documented file format independent of implementation of software
- no big concerns with efficiency (however, given the tree-shaped prior, some solutions are better than others)

XML: quick syntax tour

Basic notions:

- **XML document** is a text file in the XML format.
- Documents consists of nested **elements**.
- Boundaries of an element given by a **start tag** and an **end tags**.
- Another information associated with an element can be stored in **element attributes**.

```
<?xml version="1.0" encoding="UTF-8"?>
<my_courses>
    <course id="NPFL092">
      <name>NLP Technology</name>
      <semester>winter</semester><hours_per_week>1/2</hours_per_week>
      <department>Institute of Formal and Applied Linguistics</department>
      <teachers>
        <teacher>Rudolf Rosa</teacher>
        <teacher>Zdeněk Žabokrtský</teacher>
      </teachers>
    </course>
</my_courses>
```

XML: quick syntax tour (2)

- Tags:
  - Start tag `<element_name>`
  - End tag `</element_name>`
  - Empty `element <element_name/>`
- Elements can be embedded, but they cannot cross $\rightarrow$ XML document $=$ tree of elements
- There must be exactly one root element.
- Special symbols $<$ and $>$ must be encoded using entities (“escape sequences”) &lt; and &gt; , & $\rightarrow$ &amp;
- Attribute values must be enclosed in quotes or apostrophes; (another needed entities: &quot; and &apos;)

Time for a question

- What is the shortest length of an XML file?

XML: quick syntax tour (3)

- XML document can contain instructions for xml processor
- the most frequent instruction – a declaration header:

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

- document type declaration:

```
<!DOCTYPE MojeKniha SYSTEM "MojeKniha.DTD">
```

- Comments (not allowed inside tags, cannot contain –)

```
<!-- bla bla bla -->
```

- If the document conforms to all syntactic requirements: a **well-formed** XML document
- Well-formedness does not say anything about the content (element and attribute names, the way how elements are embedded...)
- Checking the well-formedness using the Unix command line:

```
> xmllint --noout my-xml-file.xml
```

Time for an exercise

- Use a text editor for creating an XML file, then check whether it is well formed.

Need to describe the content formally too?

- well-formedness – only conforming the basic XML syntactic rules, nothing about the content structure
- but what if you need to specify the structure
- several solutions available
    - DTD – Document Type Definition
    - other XML schema languages such as RELAX NG (REgular LAnguage for XML Next Generation) or XSD (XML Schema Definition)

DTD – Document Type Definition

DTD

- Came from SGML
- Formal set of rules for describing document structure
- Declares element names, their embeding, attribute names and values...
- example: a document consisting of a sequence of chapters, each chapter contains a title and a sequence of sections, sections contain paragraphs...

DTD location

- external DTD – a stand-off file
- internal DTD – inside the XML document

DTD Validation

- the process of checking whether a document fulfills the DTD requirements
- if OK: the document is **valid with respect to the given DTD**
- of course, only a well-formed document can be valid
- checking the validity from the command line:

```
> xmllint --noout --dtdvalid my-dtd-file.dtd  my-xml-file.xml
```

- an unfortunate terminological confusion: you can often see the term ‘validation’ or ‘validator’ used in the sense of well-formedness checking/checker

DTD structure

- Four types of declarations
- Declaration of elements <!ELEMENT ...>
- Declaration of attributes <!ATTLIST ...>
- Declaration of entities
- Declaration of notations

Declaration of elements

- Syntax: <!ELEMENT name content>
- A name must start with a letter, can contain numbers and some special symbols .-_:
- Empty element: <!ELEMENT název EMPTY>
- Element without content limitations: <!ELEMENT název ANY>

Declaration of elements (2)

- Text containing elements
  - Reserved name PCDATA (Parseable Character DATA)
  - Example: <!ELEMENT title (#PCDATA)>
- Element content description – regular expressions
- Sequence connector ,
- Alternative connector |
- Quantity ? + *
- Mixed content example: <!ELEMENT emph (#PCDATA|sub|super)* >

Declaration of attributes

- Syntax: <!ATTLIST element_name declaration_of_attributes>
- declaration of an attribute
    - attribute name
    - attribute type
    - default value (optional)
- example: <!ATTLIST author firstname CDATA surname CDATA>

Declaration of attributes (2)

- Selected types of attribute content:
  - CDATA – the value is character data
  - ID – the value is a unique id
  - IDREF – the value is the id of another element
  - IDREFS – the value is a list of other ids
  - NMTOKEN – the value is a valid XML name
  - ...
- Some optional information can be given after the type:
  - #REQUIRED – the attribute is required
  - ...

A DTD example (credit: w3schools.com)

```
<!DOCTYPE TVSCHEDULE [

<!ELEMENT TVSCHEDULE (CHANNEL+)>
<!ELEMENT CHANNEL (BANNER,DAY+)>
<!ELEMENT BANNER (#PCDATA)>
<!ELEMENT DAY (DATE,(HOLIDAY|PROGRAMSLOT+)+)>
<!ELEMENT HOLIDAY (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT PROGRAMSLOT (TIME,TITLE,DESCRIPTION?)>
<!ELEMENT TIME (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>

<!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
<!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
<!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>
<!ATTLIST TITLE RATING CDATA #IMPLIED>
<!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>
]>
```

An external DTD

- a separate file,
- could be referred from an XML file using a processing instruction:

```
<!DOCTYPE nameofmyrootelement SYSTEM "mydtdfile.dtd">
```

- DTD example (credit: w3schools.com):

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

An internal DTD included inside XML file (credit: w3schools.com)

- included into an XML file
- Example (credit: w3schools.com):

```
 <?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note><to>Tove</to><from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

Time for a question

- When would you prefer to store DTD internally and when externally?

DTD pros and cons

- positive: very simple, concise syntax
- negative: a DTD itself is not an XML file
- negative: DTD much less expressive compared to e.g. to XML Schema

Time for an exercise

- What can go wrong with an XML file if you check its well-formedness and validity. How would you check whether the requirements are fulfilled?

Criticism of XML

- quite verbose (well, you can always compress your XML files, but still)
- computationally demanding when it comes to huge data and/or limited hardware capacity
- relatively complex
- redundant
- simpler and less lengthy alternatives are popular now now such as
    - JSON – suitable for interchange of structure data
    - markdown – for textual documents with simple structure

Summary

1. XML = an easy-to-process file format
2. platform-independent
3. self-documented structure (if properly-designed)
4. thus excellent for data exchange
5. createable using any text editor, readable by naked eye
6. tree-shaped logical skeleton
7. open specification, no specialized software needed
8. a bit too verbose, not optimal if speed is an issue
9. standard libraries existing in most programming languages (next week)

https://ufal.cz/courses/npfl092